




Universitetet
i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Automatisering og signalbehandling	Vårsemesteret, 2018 Åpen / Konfidensiell
Forfatter: Roger Otto Eliassen	 (signatur forfatter)
Fagansvarlig: Karl Skretting Veileder(e): Karl Skretting	
Tittel på masteroppgaven: Utendørs semi-autonom navigasjon og rutefølgning for lite kjøretøy. Engelsk tittel: Outdoor semi-autonomous navigation and route following for small vehicle.	
Studiepoeng: 30	
Emneord: Autonomi, robot, GPS, KVS Technologies	Sidetall: 110 + vedlegg/annet: 1 Stavanger, 15. juni 2018



Universitetet
i Stavanger

Utendørs semi-autonom navigasjon og
rutefølging for lite kjøretøy

Masteroppgave
Roger Eliassen
Juni 2018

Det teknisk-naturvitenskapelige fakultet
Institutt for data- og elektroteknologi
Universitetet i Stavanger

Forord

Det rettes en takk til KVS Technologies for at de utarbeidet og foreslo en spennende og utfordrende oppgave, i tillegg til å tilby meg kontor plass i deres lokaler på Forus. De har også vært generøse med utlån av utstyr og sensorer benyttet i denne oppgaven, samt tilbudt god støtte underveis. Jeg vil også takke Universitetet i Stavanger for lån av Spurv, samt min veileder Karl Skretting for god støtte under prosjektet.

Sammendrag

I dagens samfunn benyttes roboter i økende grad til å utføre tjenester, og de fleste er forhåndsprogrammert til statiske oppgaver innen industriell produksjon. Sakte men sikkert ser man også inntog av roboter med *autonome* egenskaper. Utgangspunktet for oppgaven er selskapet KVS Technologies sitt kjøretøy kalt *Spurv*, som idag brukes til å inspisere tunnelbranner og lignende av brannvesenet. Kjøretøyet, eller roboten, som er konstruert med samme styringsgeometri som en bil har mange andre potensielle bruksområder, blant annet innen militær og industriell overvåkning.

Brukeren har frem til nå vært avhengig av en operatør for å kontrollere roboten, mens denne oppgavens hensikt er å implementere *semi-autonom* funksjonalitet for *Spurv*, slik at operatøren frigjøres fra detaljstyring av én robot, til å kunne overvåke en flåte av roboter. Det er ønskelig at det resulterende systemet skal opereres ved å spesifisere en rute, som kan inneholde ønske om inspeksjon av interessante objekter, som roboten følger selvstendig. For et slikt system er det av kritisk betydning at det kjenner egen posisjon med høy presisjon, samtidig som det er viktig at kjøretøyet kan oppdage fysiske hindringer og unngå dem.

Oppgaven bygger på Robot Operating System (ROS) og drar stor nytte av dets store og engasjerte programvare-miljø for utvikling av robotikk-systemer, og der eksisterende programvare ikke strekker til, utvikles det egne konsepter spesielt tilpasset *Spurv*.

Gjennom eksperimenter vurderes og tilpasses de opprinnelige lokaliseringssensorene (motorodometri), før det implementeres nye *relative* (IMU, visuell odometri) og *globale* (GPS-mottaker) sensorer. Formålet med de nye sensorene er å forbedre estimering av robotens posisjon ytterligere, og dermed muliggjøre semi-autonom oppførsel. I tillegg implementeres det et stereokamera som tilfører roboten maskinsyn for oppdagelse av hindringer.

Testenes resultat diskuteres og resulterer i et anbefalt oppsett bestående av en kombinasjon av IMU (estimat av vinkel) og motorodometri sammen med GNSS-mottaker (estimat av translasjon). For maskinsyn vurderes stereokamera å gi tilfredsstillende resultat for oppdagelse av hindringer.

Oppgaven avslutter med å konkludere med at det anbefalte oppsettet lykkes i å tilføre *Spurv* grunnleggende semi-autonom funksjonalitet som gjør at roboten kan benyttes til overvåkning av større områder uten detaljstyring fra en operatør. Studenter ved Universitet i Stavanger vil kunne dra nytte av rapporten og det resulterende systemet for å sette seg inn i autonome kjøretøy, samt drive videre utvikling.

Innhold

1	Introduksjon	1
1.1	Motivasjon	1
1.2	Deloppgaver	3
1.3	Rapportinndeling	4
2	Bakgrunn	5
2.1	Spurv	5
2.1.1	Plattform	8
2.1.2	Prosessering	8
2.1.3	Kommunikasjon	8
2.1.4	Fremdrift	9
2.2	Sensorer	13
2.2.1	Sensorer for lokalisering	13
2.2.2	Sensor for maskinsyn	20
2.3	Robot Operating System (ROS)	23
2.3.1	Oppbygging	23
2.3.2	Selvstendig navigasjon	25
2.3.3	Referansekoordinatsystem	30
2.3.4	Konvensjoner for koordinatsystem og enheter	33
2.4	Bakgrunn for sentrale noder lagt til i ROS	36
2.4.1	Sensorfusjon	36
2.4.2	Hastighetskontroll	38
2.5	Globale koordinatsystem	39
2.5.1	WGS 84	39
2.5.2	Koordinatsystem	39
3	Konstruksjon av forbedret navigasjonssystem	43
3.1	Systemets oppbygging	43

3.1.1	Rutefølgning med oppgaver	43
3.1.2	ROS-meldinger	47
3.1.3	Sensorfusjon	48
3.2	Modifisering av eksisterende noder	49
3.2.1	IMU driver	49
3.2.2	Konvertering av dybdebilde til laserscan	50
3.3	Parametre med særlig betydning	51
3.3.1	Navigasjon	51
3.3.2	Ytelse og beregningsmessig belastning	53
4	Eksperimenter og resultater	58
4.1	Evaluering av odometri: translasjon	58
4.1.1	Testoppsett	58
4.1.2	Resultat, translasjon	60
4.2	Evaluering av odometri: sving	64
4.2.1	Testoppsett	64
4.2.2	Resultat	66
4.3	Navigasjon ved relativ posisjonering	70
4.3.1	Testoppsett	70
4.3.2	Resultat	73
4.4	Navigasjon ved global posisjonering	79
4.4.1	Testoppsett	79
4.4.2	Resultat	81
4.5	Modellering av omgivelser	87
4.5.1	Testoppsett	87
4.5.2	Resultat	88
5	Diskusjon med konklusjon	90
5.1	Evaluering av odometri	90

5.1.1	Relative sensorers presisjon	90
5.2	Navigasjon	92
5.2.1	Navigasjon ved relativ posisjonering	92
5.2.2	Navigasjon ved global posisjonering	93
5.2.3	Deteksjon og unngåelse av hindringer	95
5.3	Anbefalt systemoppsett	97
5.4	Videre arbeid og forbedringer	97
5.5	Konklusjon	99
6	Referanser	100
7	Vedlegg	103

Ordliste

- EKF** - Extended Kalman Filter (*norsk: utvidet Kalman-filter*)
- ENU** - East North Up (*Øst Nord Opp*)
- ERPM** - Electrical Revolutions Per Minute, (*norsk: elektriske omdreininger per minutt*)
- GNSS** - Global Navigation Satellite System (*norsk: globalt satellitt-navigasjonssystem*)
- GPS** - Global Positioning System (*norsk: globalt posisjoneringssystem*)
- HDOP** - Horizontal Dilution Of Precision *horisontal forringelse av presisjon*
- IMU** - Inertial Measurement Unit (*Treghetsmålingsenhet*)
- NED** - North East Down (Nord Øst Ned)
- REP** - ROS Enhancement Proposals (*ROS forbedringsforslag*)
- ROS** - Robot Operating System
- RTK** - Real Time Kinematic *sanntid kinematisk*
- UKF** - Unscented Kalman Filter
- UTM** - Universal Transverse Mercator
- WGS 84** - World Geodetic System 1984

1 Introduksjon

1.1 Motivasjon

I dagens samfunn benyttes roboter i økende grad til å utføre tjenester, og de fleste er forhåndsprogrammert til statiske oppgaver innen industriell produksjon. Sakte men sikkert ser man også inntog av roboter med *autonome* egenskaper. Det finnes utallige definisjoner på hva som definerer en *autonom robot*, ordet i seg selv betyr *uavhengig*, som i denne konteksten kan sies å bety *menneskelig* uavhengighet. Enkelte aspekter forekommer likevel hyppig:

- Evnen til å hente inn og behandle informasjon om omgivelser; *maskinsyn*.
- Evnen til å operere selvstendig over et lengre tidsrom uten menneskelig påvirkning; *gjenoppretter seg fra feil*.
- Evnen til å bevege seg selv uten menneskelig assistanse, herunder evnen til å nøyaktig beregne egen posisjon; *selvstendig navigasjon og lokalisering*.
- Evnen til å unngå situasjoner som er skadelig for mennesker, materiell og seg selv; *unngåelse av hindringer*.

Det legges spesielt merke til én ting som er felles for alle punktene i listen over; de avhenger av at roboten kjenner egen posisjon, med så høy nøyaktighet som mulig.

Videre kan en robots *grad* av autonomi deles opp i følgende tre klasser:[1]

- **Fullt autonom:** En fullt autonom robot er i stand til å løse komplekse oppgaver på egenhånd, og har egne *intensjoner*. Den har også evnen til å selvstendig planlegge handlinger basert på innhentet informasjon.
- **Semi-autonom:** En semi-autonom robot er til en viss grad eksternt kontrollert, men kan også utføre handlinger basert på innhentet informasjon. Et godt eksempel er en robot som krever at brukeren spesifiserer *hvor* den skal, mens den kan finne veien selv og unngå hindringer underveis.
- **Ikke-autonom:** En ikke-autonom robot er fullstendig kontrollert av en ekstern aktør, og har ingen *egne* intensjoner og benytter enkel styringslogikk.

Med definisjonene over i bakhånd presenteres utgangspunktet for denne oppgaven; roboten "Spurv". Spurv er et lite kjøretøy som er 0.5 m lang og 0.45 m bred som er bygget med samme styringsgeometri som en bil . Den er utviklet av det lokale selskapet "KVS Technologies", og benyttes idag av brannvesenet til inspeksjon av tunnelbranner og andre scenarier som innebærer stor risiko ved menneskelig interaksjon. Det er en *ikke-autonom* robot som er avhengig av en ekstern operatør for

kontroll via en trådløs tilkobling, og inneholder to kamera, enkel funksjonalitet for å beregne egen posisjon i tillegg til kapabel prosesseringskraft som på langt nær er fullt utnyttet.

Slik funksjonalitet, kombinert med en ypperlig fysisk plattform, gjør at roboten har mange andre potensielle bruksområder, blant annet innen militær og industriell overvåkning av større områder. Ved å benytte ikke-autonome roboter til et slikt formål, vil hver enkelt robot kreve kontinuerlig oppmerksomhet av en operatør. Hvis derimot en (semi-)autonom robot benyttes, vil en enkelt operatørs rolle kunne gå fra detaljstyring til overvåking. Operatøren vil også ha kapasitet til å overvåke en *flåte* av roboter istedenfor kun én enkelt robot.

Denne oppgavens hensikt er å implementere semi-autonom funksjonalitet for Spurv, slik at den kan brukes til å løse oppdrag som går ut på, men ikke begrenses til, planlagt overvåkning av større områder med bebyggelse. Det er ønskelig at det resulterende systemet skal opereres ved å spesifisere en rute, som kan inneholde ønske om inspeksjon av interessante objekter, som roboten deretter følger selvstendig. Roboten må også være i stand til å oppdage hindringer underveis og unngå dem ved nødvendighet. Implementasjonen forsøkes å gjøres generell, slik at løsningen ikke knyttes spesifikt til Spurv, men muliggjør videreføring av konsept til større kjøretøy som benytter det samme konstruksjonsprinsippet.

For realisering av oppgavens mål, deles problemstillingen opp i deloppgaver.

1.2 Deloppgaver

Oppgaven har fire hovedpunkt som ønskes løst. Nedenfor er hvert av disse listet med beskrivelse for hvert punkt.

Forbedring av posisjonsnøyaktighet

En grunnleggende og svært viktig funksjon for en semi-autonom robot er evnen til å estimere egen posisjon med høy presisjon over tid. Roboten har fra før en sensor for beregning av posisjon, men den antas å ikke kunne dekke behovet for denne oppgaven alene. Det skal derfor implementeres og testes flere sensorer for å forbedre robotens evne til å *lokalisere* seg selv.

Før sensorene kan bidra må de kalibreres og tilpasses roboten, og det må finnes en metode for å *kombinere* ulike sensorers målinger.

Rutefølging

Roboten skal ha muligheten til å følge en rute spesifisert av bruker. For å oppnå slik funksjonalitet må det etableres et felles *koordinatsystem*, slik at brukers forventning og robotens forståelse stemmer overens. Videre må posisjonsangivelse fra bruker føre til at roboten beveger seg mot målet, det må altså implementeres et system for *selvstendig* navigasjon. Slik funksjonalitet finnes i programvaren ROS (Robot Operating System), som konfigureres og videreutvikles for det spesifikke oppsettet i denne oppgaven.

Deteksjon og unngåelse av hindringer

Under rutefølging må roboten være i stand til å detektere hindringer slik at det kan ta nødvendige forholdsregler for å unngå kollisjon. Ved bruk av en optisk sensor som gir roboten *maskinsyn*, kan den også ta vare på oppdagede hindringer og unngå dem ved neste anledning.

Oppgaver

Et naturlig steg videre er muligheten til å utføre spesifikke oppgaver underveis i ruten. Eksempelvis kan en slik oppgave være å stanse ved bestemte lokasjoner i et viss tidsrom mens et spesifikt objekt eller bygning inspiseres ved bruk av videokamera, før den drar videre til neste lokasjon.

1.3 Rapportinndeling

Rapporten deles opp i fire kapitler:

Bakgrunn

Her presenteres teorien bak programvaren som er benyttet, matematikken som danner grunnlaget for databehandling og kalkulasjoner samt den fysiske oppbyggingen av roboten. Her beskrives også eksisterende biblioteker og programvare som er tatt i bruk.

Konstruksjon

Kapitlet presenterer og beskriver egenutviklet programvare samt nødvendige endringer i programmer utviklet av andre. Systemets oppbygging og funksjon beskrives, før leseren settes inn i betydningen av de forskjellige parametrene som inngår i prosjektet.

Eksperimenter og resultater

Eksperimenter som er utført for prosjektet beskrives og resultatene presenteres. De første eksperimentene omhandler grunnleggende konsepter som er kritiske for systemets funksjonalitet, før det gradvis går over i mer kompleks oppførsel. Kapitlet presenterer data som danner grunnlaget for valg gjort underveis, og som ender opp i et anbefalt systemoppsett.

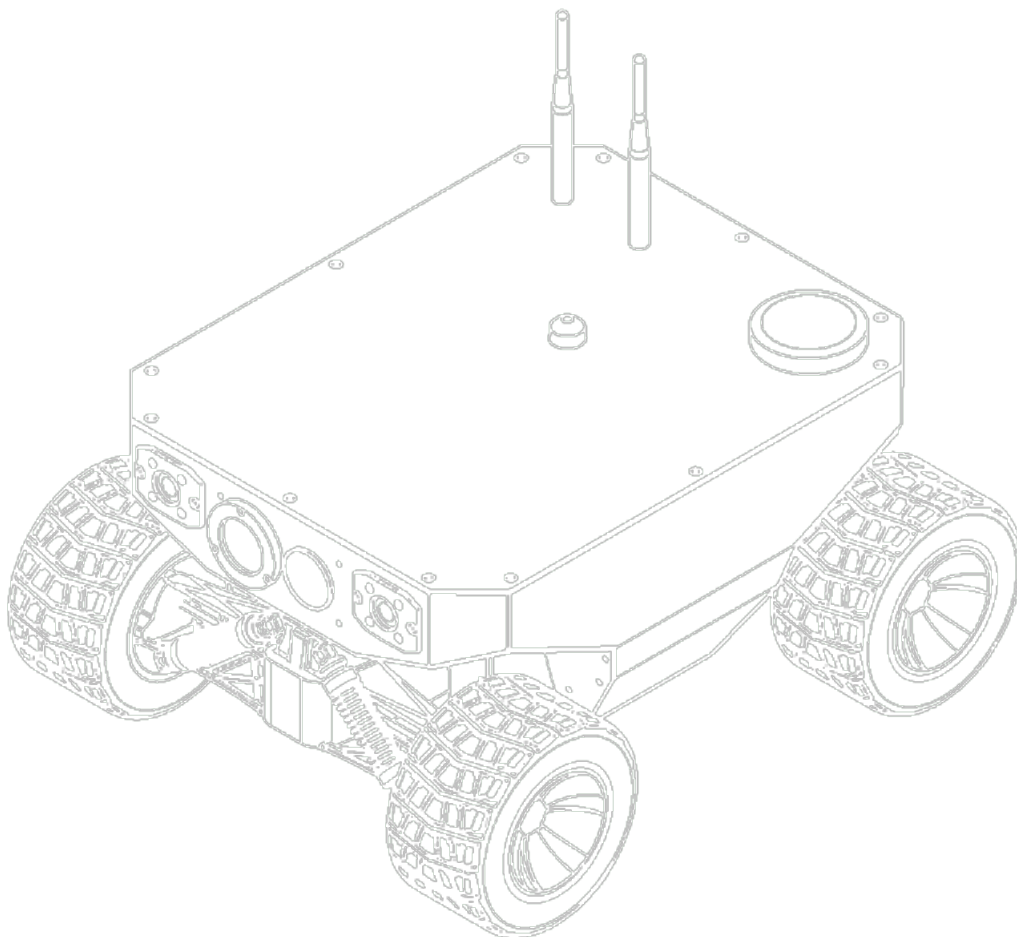
Diskusjon og konklusjon

Her sammenlignes resultatene med deloppgavene definert i introduksjonen. De ulike sensorkonfigurasjonenes styrker og svakheter diskuteres, og kapitlet rundes av med anbefalt systemoppsett. Deretter utredes forslag til forbedringer og videre arbeid før konklusjonen avslutter oppgaven.

2 Bakgrunn

Dette kapitlet inneholder relevant bakgrunnsstoff nødvendig for å kunne sette seg inn i problemstilling og løsning for denne oppgaven.

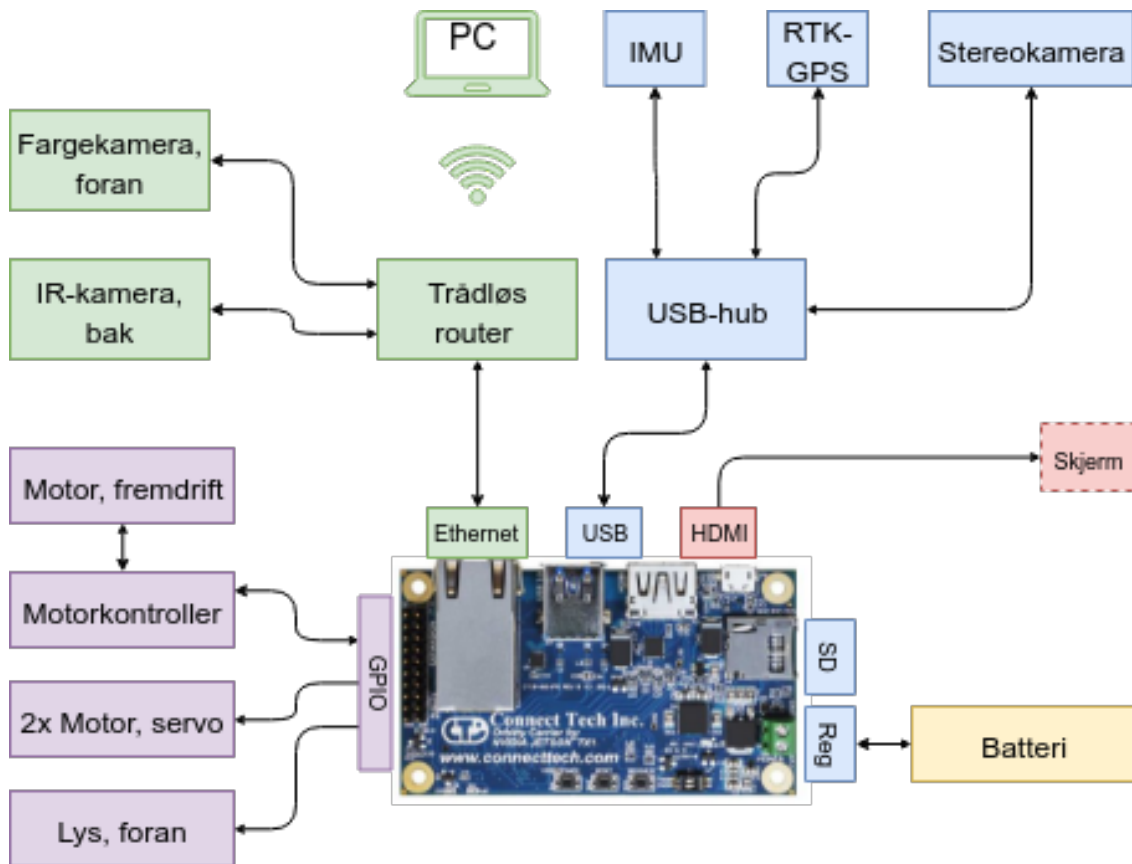
2.1 Spurv



Figur 1: Konseptskisse, Spurv. Størrelser: lengde: 500 mm, bredde: 450 mm, høyde: 250 mm.

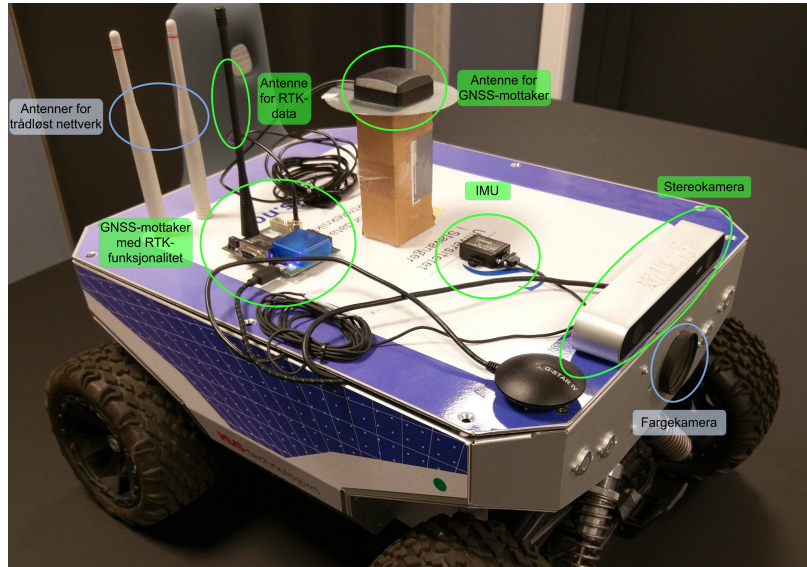
Dimensjoner	Lengde	500 mm
	Bredde	450 mm
	Høyde	250 mm
Vekt		10 kg
Hjulradius		15 cm
Makshastighet		12 m/s
Motor	Fremdrift	Børsteløs DC-motor
	Sving	2 x analog servomotor
	Kontroller	Feltorientert styring
Sensorer	Maskinsyn	Zed Labs Stereokamera
	Odometri	Magnetisk enkoder, IMU, RTK GPS, visuell odometri
	Annet	Fargekamera (foran), termisk kamera (bak)
Kommunikasjon		WiFi 5 GHz
Prosessering	Modell	Nvidia Jetson Tegra TX2
	CPU	4-kjerners ARM @ 2.0 GHz + 2-kjerners Denver @ 2.0GHz
	GPU	256-kjerners Pascal @ 1.3 GHz
Strømforsyning	Batteri	Oppladbart Litium Polymer
	Operasjonstid	>3 timer

Tabell 1: Nøkkeldata, Spurv



Figur 2: Oversikt, maskinvare. Kretskortet som kobler sammen alle enhetene er et utvidelseskort for prosesseringsenheten som utvider antall tilkoblinger. Det kan kobles til en skjerm hvis ønskelig, for utvikling og feilsøking.

2.1.1 Plattform



Figur 3: Figuren viser en oversikt over *Spurv* og tilleggsmonterte sensorer (grønt). Blått er fastmontert utstyr.

Roboten er bygget på en solid plastramme med kraftige gummihjul som stikker ut fra skroget både foran, bak og på siden for beskyttelse. Skroget er konstruert i aluminium og huser motorer, batteri og annen maskinvare. På baksiden har den tilkoblinger for ethernet, USB og HDMI, samt en strømbryter og ladeplugg.

2.1.2 Prosessering

Prosesseringsenheten i *Spurv* er en plattform fra Nvidia kalt **Jetson TX2** som inneholder blant annet CPU, GPU, RAM og lagringsplass på et enkelt kort med størrelse 50 x 87 mm og vekt på 85 gram. På lagringsenheten er det installert operativsystemet Ubuntu. Av tilkoblinger har den USB 3.0 og gigabit ethernet. Det er et kort spesielt utviklet for mobile enheter og leverer dermed god ytelse ved lavt strømforbruk. Typisk forbruk ligger på 7.5 W (625 mA @ 12 V) med en maksimum på ca 15 W (1.25 A @ 12V). [2]

2.1.3 Kommunikasjon

Ekstern kommunikasjon

Roboten inneholder en egen trådløs router som gjør at den enten kan koble seg på eksisterende trådløse nett eller sette opp et selvstendig nettverk. Bruk av standardisert nettverksteknologi gjør at det er enkelt å kommunisere med *Spurv* og tillater

den å sende store mengder data på direkten til bruker for analyse under utvikling og ved kjøring. Router opererer utelukkende i 5 GHz-båndet.

Det er også mulig å konfigurere router med 4G teknologi, som gjør at roboten kan drar nytte av et geografisk utbredt nettverk.

Intern kommunikasjon

Internt kommuniserer de ulike komponentene hovedsaklig over USB og ethernet. En fullstendig oversikt kan sees i figur 2.

2.1.4 Fremdrift

Motor, fremdrift

Spurv benytter en børsteløs likespenningsmotor til fremdrift. Denne motortypen har et høyt kraft-til-vekt forhold og har den fordelen at den kan kontrolleres elektronisk, som gir høy presisjon og mange konfigurasjonsmuligheter. Motoren, som kan yte opp mot 1000 W, driver alle fire hjul gjennom en drivaksel. Den kan styres på grunnlag av omdreiningshastighet, strømstyrke eller kraft. Den har ingen fysiske sensorer for å holde kontroll på rotorposisjon, men drar nytte av det elektriske fenomenet *motelektromotorisk spenning* for å oppnå tilbakemelding om reell hastighet. Motorens omdreiningshastighet representeres av størrelsen *electrical revolutions per minute*, norsk: *elektriske omdreininger per minutt*, som har følgende forhold til den mer kjente størrelsen *rpm* ved:

$$erpm = \text{antall polpar} * rpm \quad (1)$$

Motor, sving

Roboten benytter samme styringsgeometri som en vanlig bil, og svinger dermed med fronthjulene. Svingvinkel kontrolleres av to servomotorer som sammen kontrollerer sving for begge hjulene. De inneholder en enkel proporsjonal-regulator som sørger for at utslagsvinkel er lineær med påtrykt spenning, men gir ingen form for tilbakemelding.

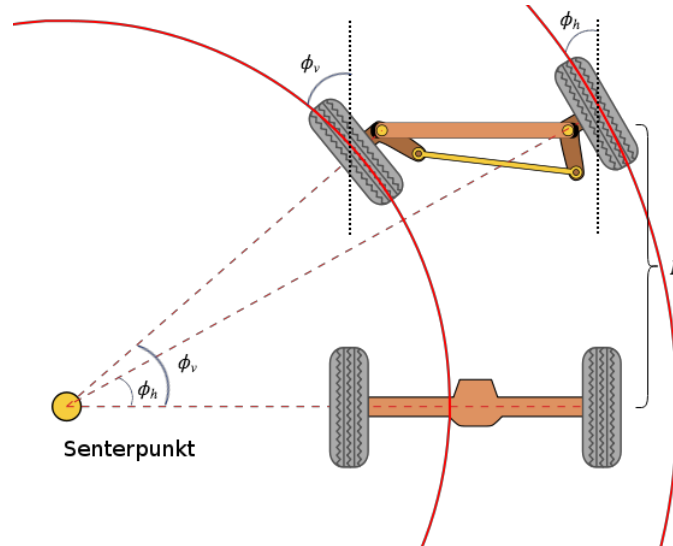
Motorkontroller

Motorkontrollerens oppgave er å konvertere en hastighetskommando til en tilsvarende motorspenning som driver motoren i ønsket hastighet. Siden motelektromotorisk spenning er proporsjonal med motorens rotasjonshastighet kan den brukes til å beregne hastighet og dermed estimat av tilbakelagt distanse.

Styringsgeometri

Mange roboter benytter andre kinematiske modeller, som ofte gjør at de kan svinge rundt sin egen akse. Én slik modell kalles *differensiell* styring. Spurv kan ikke rotere

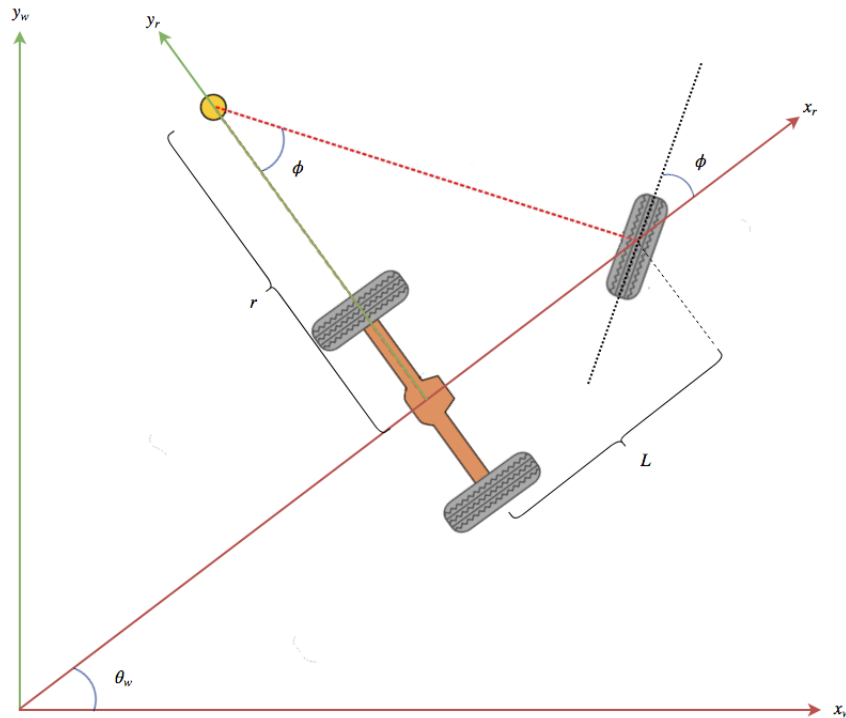
om sin egen akse, og når en slik form for styring benyttes må den fysiske koblingen mellom hjulene konstrueres på en spesiell måte for å fungere som ønsket. Det kalles *Ackermann-styring* og er illustrert i figur 4.



Figur 4: Konseptet bak Ackermann-styring, med senterpunkt og styrevinklene $\phi_v > \phi_h$ for hhv. venstre og høyre hjul. Styrevinklene finnes også igjen ut fra svingradier og senterpunkt. L er kjøretøyets hjulavstand.

Hensikten med å ta i bruk en slik styringsmetode er å løse problemet som oppstår ved at det indre og ytre hjulet må ha forskjellig vinkelutslag ved sving. Et kjøretøy som svinger på forhjulene, men som ikke er konstruert etter denne metoden vil oppnå samme vinkel for begge hjul, som resulterer i at det innerste hjulet vil skli sideveis og dermed vil ikke kjøretøyet klare å gjennomføre den ønskede svingen. Konstrueres derimot koblingen mellom hjulene etter Ackermanns styringsgeometri ruller alle hjulene langs en sirkel med forskjellig radius, men felles senterpunkt. Det sørger for at det innerste hjulet får en skarpere sving og dermed unngås problematikken med hjul som dras sidelengs.

Foroverkinematikk



Figur 5: Approksimering av ackermann styringsgeometri, ϕ representerer resulterende svingvinkel for ett hjul. xy_w representerer strekning i verdenskoordinater, mens xy_r representerer strekning i robotens lokale koordinatsystem. L er robotens hjulavstand mens r er svingradius.

Ligningene som beskriver robotens foroverkinematikk brukes til å estimere posisjon basert på motorens omdreiningshastighet og hjulenes svingvinkel. Siden ackermanngeometrien sørger for at alle hjul kjører rundt et felles senterpunkt, kan dynamikken tilnærmes ved å modellere den som en trehjulssykkel som vist i figur 5. Her beskrives forholdet mellom verdenskoordinater, xy_w , og robotens koordinatsystem, xy_r .

For å uttrykke verdenskoordinater som funksjon av x_r benyttes figur 5 som utgangspunkt.

Ved bruk av trigonometri finnes robotens posisjon i verdenskoordinater ved

$$x_w = x_r \cos(\theta_w)$$

og

$$y_w = x_r \sin(\theta_w)$$

Deretter finnes hastighet i verdenskoordinater ved derivasjon

$$\dot{x}_w = \dot{x}_r \cos(\theta_w) \quad (2)$$

$$\dot{y}_w = \dot{x}_r \sin(\theta_w) \quad (3)$$

Roboten kan ikke bevege seg sidelengs som medfører at $\dot{y}_r = 0$.

For å finne vinkelhastighet $\dot{\theta}_w$ tas det utgangspunkt i definisjonen for *tangensiell hastighet*:

$$v_{\perp} = r \frac{d\theta}{dt}$$

som omskrevet til gjeldende notasjon gir

$$\dot{x}_r = r \dot{\theta}_w \quad (4)$$

Figur 5 viser at styrevinkel ϕ kan beskrives ved forholdet

$$\tan(\phi) = \frac{L}{r} \quad (5)$$

Ved å sette ligning 4 inn i 5 kommer uttrykket for vinkelhastighet i verdenskoordinater frem:

$$\dot{\theta}_w = \frac{\dot{x}_r}{r} = \frac{\dot{x}_r}{\frac{L}{\tan(\phi)}} = \dot{x}_r \frac{\tan(\phi_w)}{L} \quad (6)$$

Siden tilbakemelding fra fremdriftsmotoren kommer i form av \dot{x}_r , og utslag som ϕ , kan kinematikklikningene 2, 3 og 6 brukes direkte for å finne tilbakelagt strekning i verdenskoordinater ved integrasjon:

Først finnes vinkelen slik at den kan settes inn i likningene for strekning

$$\theta_w = \int_0^T \dot{\theta}_w dt = \int_0^T \frac{\tan(\phi)}{L} \dot{x}_r dt \quad (7)$$

Deretter settes vinkelen inn i ligninger for strekning, som tilslutt gir strekning i verdenskoordinater.

$$x_w = \int_0^T \dot{x}_w dt = \int_0^T \dot{x}_r \cos(\theta_w) dt \quad (8)$$

$$y_w = \int_0^T \dot{y}_w dt = \int_0^T \dot{x}_r \sin(\theta_w) dt \quad (9)$$

2.2 Sensorer

Kapitlet beskriver sensorene som er benyttet i denne oppgaven, og beskriver både sensorer for beregne posisjon samt sensor for maskinsyn. Alle sensorer utenom motorodometri er implementert for denne oppgaven.

2.2.1 Sensorer for lokalisering

Systemet benytter flere sensortyper som sammen kan produsere et presist estimat av posisjon. Sensorene deles opp i to hovedgrupper: *relative* og *globale* sensorer. En *relativ* sensor beskriver nåværende posisjon *relativt* til en posisjon ved et tidligere tidspunkt, mens en *global* sensor produserer en *absolutt* posisjon. Det følgende underkapitlet går igjennom sensorene benyttet i denne oppgaven.

Motorodometri

Motorodometri er et posisjonsestimat som baseres på motorens omdreiningshastighet og servomotorenes utslag. Omdreiningshastighet rapporteres av motorkontroller basert på måling av motelektromotorisk kraft (EMF), mens for servomotorenes utslag må det brukes *ønsket* styrevinkel. Det er en potensiell svakhet for estimat av vinkel, siden det ikke er basert på faktiske målinger, men istedet lener seg på antakelsen om at satt verdi stemmer overens med reell verdi. Før størrelsene kan brukes i kalkulasjoner må de først konverteres til sine respektive SI-enheter ved hjelp av konverteringskonstantene `steering_to_servo_gain` og `speed_to_erpm_gain`.

Robotens hastighet, \dot{x}_r [m/s], estimeres fra motorens rapporterte omdreiningshastighet, ω_{motor} , og en omregningskonstant, $K_{SpeedToRpm}$, som beskriver forholdet mellom hastighet i *erpm* og *m/s*:

$$\dot{x}_r \text{ [m/s]} = \frac{\omega_{motor} \text{ [rpm]}}{K_{SpeedToRpm} \text{ [rpm/m/s]}} \quad (10)$$

Robotens svingvinkel, ϕ , estimeres fra ønsket servomotorutslag, $\phi_{ServoCommand}$, og en konstant, $K_{SteeringToServo}$, som beskriver forholdet mellom servomotorens utslag og svingvinkel for hjulene. Servomotorens utslag representeres av et desimaltall begrenset til å ligge innenfor servomotorens utslagsområde, som finnes mellom 0 og 1. En annen konstant, $Offset_{SteeringToServo}$, bestemmer servomotorenes nullposisjon:

$$\phi \text{ [rad]} = \frac{\phi_{ServoCommand} \text{ [utslag]} - Offset_{SteeringToServo} \text{ [utslag]}}{K_{SteeringToServo} \text{ [utslag/rad]}} \quad (11)$$

Programvaren tilknyttet motorkontroller produserer så et estimat av robotens hastighet i xy_w -planet ved å implementere ligningene 7 - 9 i diskret tid.

Et slikt posisjonsestimat er sårbart for ukjente ytre påvirkninger som hjulspinn, måleusikkerhet og andre faktorer som dynamikken mellom servomotor og hjul. Slike

feil vil manifestere seg i form av drift i posisjonsestimater over tid. Kapittel 2.4.1: 'Sensorfusjon' beskriver hvordan drift forbundet med *relative* posisjonskilder kan kompenseres for ved hjelp av metoder som *sensorfusjon* og bruk av *globale posisjonskilder*.

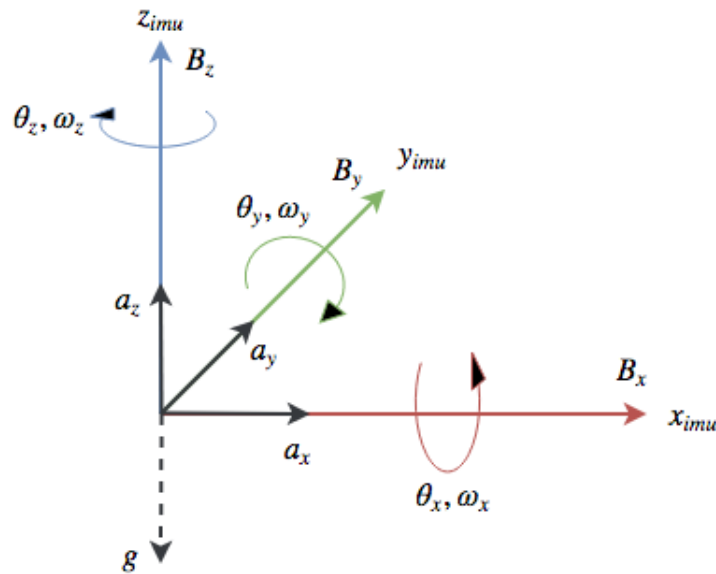
- Sensorens benyttede bidrag til lokalisering
 - xy_w : Robotens relative translasjon langs x- og y-akse, forankret i verdenskoordinater.
 - θ_z Robotens rotasjon om z-aksen.

IMU

IMU er en forkortelse for *Inertial Measurement Unit*, norsk: *treghetsmålingsenhet*. IMU er et ord som på folkemunne brukes noe unøyaktig om lignende enheter, hovedsaklig finnes det to typer: IMU og AHRS som står for *Attitude Heading Reference System*, norsk: *Positur og retning referansesystem*. Med ordet positur menes enhetens translasjon og rotasjon i rommet, \mathbb{R}^3 .

En IMU inneholder en samling av måleinstrumenter som hver for seg måler de fysiske størrelsene akselerasjon [m/s^2], vinkelhastighet [rad/s] og magnetisk flukstetthet [G]. Instrumentene kalles respektivt akselerometer, gyroskop og magnetometere. Et slikt system måler alle størrelser i de 3 dimensjonene x, y og z, som tilsammen gir 9 målinger, illustrert i figur 6.

En AHRS inneholder det samme som en IMU, men har i tillegg mulighet til å prosessere og filtrere målingene. Dette gjør at den kan produsere absolutt rotasjon[rad] i motsetning til en IMU som kun gir rotasjonshastighet[rad\s]. Denne oppgaven benytter en AHRS, men resten av rapporten refererer til den som IMU for bedre lesbarhet.



Figur 6: De fysiske størrelsene som måles i en IMU, hvor ω er vinkelhastighet, a er akselerasjon og B er magnetisk flukstetthet. I tillegg er tyngdekraftens akselerasjon indikert med symbolet $g[m/s^2]$.

Nytteverdien i en IMU er at dens målinger kan brukes til å estimere translasjon, x_{imu} , og rotasjon, θ_{imu} , for en robot i bevegelse i forhold til et utgangspunkt, som gjerne er robotens posisjon ved oppstart. Siden akselerasjon, \mathbf{a}_{imu} , og vinkelhastighet, $\boldsymbol{\omega}_{imu}$, ikke direkte gir avstand og vinkel, må forflytning og vinkel kalkuleres, som vist for én dimensjon i ligning 12 - 11

Strekning er hastighet integrert

$$x_{imu}(t) = \int_{t=0}^T \dot{x}_{imu}(t) dt \quad (12)$$

videre er hastighet akselerasjon integrert

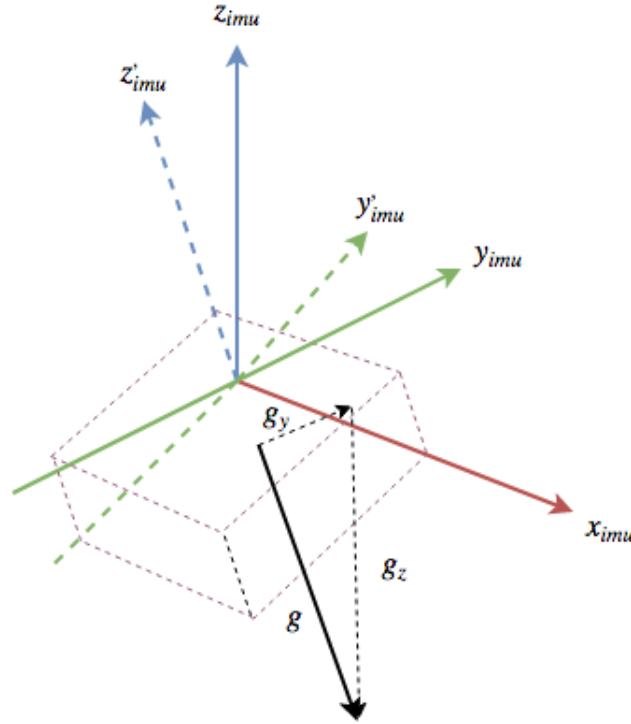
$$\dot{x}_{imu}(t) = \dot{x}_0 + \int_{\tau=0}^T a_{imu}(\tau) d\tau \quad (13)$$

13 satt inn i 12 resulterer i et dobbeltintegral for å finne strekning

$$\dot{x}_{imu}(t) = \int_{t=0}^T \left(\dot{x}_0 + \int_{\tau=0}^T a_{imu}(\tau) d\tau \right) dt \quad (14)$$

For å regne ut vinkel integreres vinkelhastighet

$$\theta_{imu}(t) = \int_{t=0}^T \omega_{imu}(t) dt \quad (15)$$



Figur 7: Figuren viser IMU rotert ca 20° om x -aksen, og hvordan tyngdekraftens akselerasjon, g , påvirker de andre dimensjonene. Videre fremheves viktigheten av god presisjon for måling av rotasjon for å identifisere bidraget.

Det er ikke helt uproblematisk å benytte måling av akselerasjon til å estimere tilbaketilbakelagt avstand. Figur 7 viser en IMU som står i ro, rotert 20° om sin egen x -akse. Her kommer det fram at den måler bidraget fra tyngdekraften som alltid virker på IMU, i tillegg til den fysiske akselerasjonen, \mathbf{a}_{fys} som oppstår ved endring i hastighet. Målingen fra IMU kan modelleres som

$$\mathbf{a}_{imu} = \mathbf{a}_{fys} - \mathbf{R}_g^{fys} \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \quad (16)$$

hvor matrisen \mathbf{R}_g^{fys} beskriver rotasjon mellom IMU og tyngdekraftens retning, mens g er tyngdekraftens størrelse.

For at målingene skal kunne brukes til å estimere posisjon må bidraget fra tyngdekraften, som ved rotasjon påvirker andre dimensjoner enn z , fjernes. Det gjøres ved å løse mhp. \mathbf{a}_{fys}

$$\mathbf{a}_{fys} = \mathbf{a}_{imu} + \mathbf{R}_g^{fys} \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \quad (17)$$

og dermed sette inn rotasjon om x-aksen på matriseform

$$\mathbf{R}_g^{fys} = \mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \quad (18)$$

Fysisk akselerasjon kan så beregnes ved:

$$\mathbf{a}_{fys} = \mathbf{a}_{imu} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} + \begin{pmatrix} 0 \\ -g \sin \theta \\ g \cos \theta \end{pmatrix} = \begin{pmatrix} a_x \\ a_y - g \sin \theta \\ a_z + g \cos \theta \end{pmatrix}$$

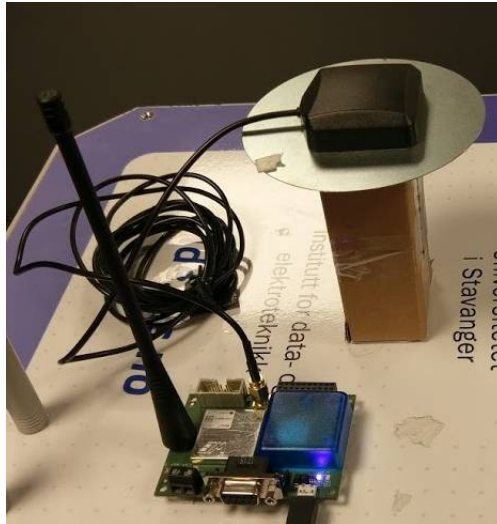
Utrekningen viser at tyngdekraften påvirker fysisk akselerasjon langs y- og z-retning avhengig av rotasjonsvinkelen θ . For å kompensere for tyngdekraftens påvirkning på målinger, er det derfor avgjørende å ha god kontroll på enhetens rotasjon om alle aksene. Denne nødvendigheten gjør at feilmarginen på rotasjonsestimering er en betydelig svakhet for en IMU. Tabellen under viser hvor stor påvirkning vinkelfeil har på de ulike størrelsene.

Vinkelfeil [°]	Akselerasjonsfeil [m/s ²]	Hastighetsfeil [m/s] etter 10s	Posisjonsfeil [m] etter 10s
0.1	0.017	0.17	1.7
0.5	0.086	0.86	8.6
1.0	0.17	1.7	17

Tabell 2: Vinkelfeilens forplantning over størrelser og tid [28]

- Sensorens benyttede bidrag til lokalisering
 - a_r : Akselerasjon i tre dimensjoner, forankret i robotens koordinatsystem.
 - θ : Estimert av robotens rotasjon i tre dimensjoner.
 - B_w : Retning for jordens magnetiske felt. Benyttes til å bestemme orientering relativt til himmelretning.

GNSS-mottaker



Figur 8: Figuren viser mottaker for satellittsignaler (oppe, høyre), selve mottaker som utfører prosessering av signaler (midten, nede), med påmontert UHF-antenne for sending av RTK-korreksjonsdata.

GNSS er forkortelse for *Global Navigation Satellite System*, norsk: *globalt navigasjons-satellittsystem*. Det er et generelt begrep som omfatter alle satellittnavigasjonssystemer, eksempelvis det mest kjente systemet *GPS*, *Global Positioning System*, norsk: *globalt posisjoneringssystem*. Mottakeren brukt i denne oppgaven kan ta imot signaler fra flere slike systemer; GPS (USA), GLONASS (Russland) og BeiDou (Kina).

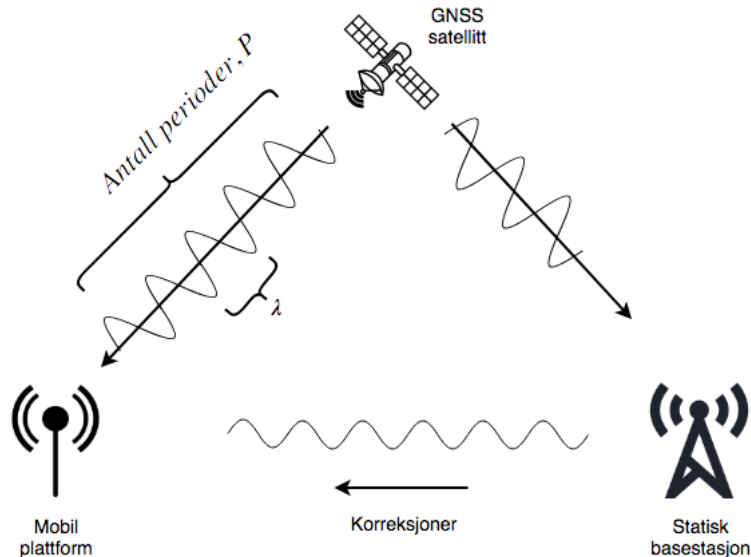
GNSS-mottakeren bruker signaler fra satellitter til å beregne egen posisjon i et forhåndsdefinert koordinatsystem. Det er definert i standarden kalt *WGS 84*, som senere beskrives mer detaljert i kapitlet 2.5. Ved beregning av posisjon kalkulerer mottakeren også posisjonestimats presisjon, som baseres på geometrien mellom seg selv og synlige satellitter, og kalles *HDOP*, *Horizontal Dilution Of Precision*, som oversettes til *horisontal degradering av presisjon*. For å regne ut usikkerhet i m^2 brukes følgende uttrykk for x- og y-retning:

$$\text{covar}([x \ y]) = (\text{hdop} \times \sigma_{[x \ y]})^2 \quad (19)$$

hvor σ er et forhåndsdefinert standardavvik for målingen, som velges basert på mottakertype. For en typisk GNSS-mottaker ligger usikkerhet (eller nøyaktighet) normalt mellom $1\text{-}3 \text{ m}^2$. Verdien kommer til nytte ved kombinasjon av data fra flere sensorer slik at de ulike estimatene vektet etter hvor gode de er. Verdien er også nyttig som et mål på hvor god dekning mottakeren har. [6]

I tillegg til normal funksjonalitet for satellittnavigasjon, benytter mottakeren også en teknologi kalt *Real Time Kinematic* (norsk: *sanntid kinematisk*). Hensikten med utvidelsen er å forbedre posisjonsestimats presisjon, og med denne funksjonen

aktivert kan mottakeren oppnå en teoretisk nøyaktighet i størrelsesorden et par centimeter. Figur 9 viser konseptet for *RTK*.



Figur 9: Konseptskisse GNSS med RTK. Avstand mellom satellitt og mobil plattform kalkuleres ved å telle antall perioder i bærebølgen, P , og multiplisere med bølgelengde, λ . Basestasjonen sender korreksjonsdata basert på presise målinger av kjente feilkilder over tid.

Et minimum RTK-oppsett består av én *basestasjon* og én *mobil enhet*. Basestasjonen plasseres slik at det er fri sikt mellom den og mobil enhet, og vil sende korreksjoner så lenge de er innenfor rekkevidde av hverandre.

Enkelt forklart beregnes den forbedrede posisjonen basert på telling av perioder, P , i satellitten(es) bærebølge, uten å benytte informasjonen modulert i signalet. Antall perioder multiplisert med bærebølgens bølgelengde, λ , gir avstand til satellitten(e). For at denne metoden skal kunne brukes er den mobile enheten avhengig av at det gjøres nøyaktige målinger i basestasjon av kjente feilkilder i signalet, som sendes til den mobile enheten. Korreksjonene sendes og mottas med en dedikert antenne som opererer i UHF-båndet (433 MHz) med en overføringshastighet på 48 kbps.[7] [8]

De to enhetene er identisk bygget opp og konfigureres for hver rolle. Det finnes flere måter å konfigurere basestasjonen på, i denne oppgaven er det benyttet en modus kalt *survey-in* som beregner mottakerens posisjon med større nøyaktighet jo lenger tid tilgjengelig. Det tar ca 5 timer å oppnå en presisjon på 0.26 m, som er benyttet i denne oppgaven.

Ved beregning av kovarians benyttes det en lavere verdi for σ for å reflektere den økte presisjonen som oppstår uavhengig av geometrien mellom mottaker og satellitter.

- Sensorens benyttede bidrag til lokalisering

- xy_w : Robotens absolutte translasjon langs x- og y-akse i verdenskoordinater.

Visuell odometri

Stereokamera har funksjonalitet som gjør at det kan benyttes til lokalisering. Konseptet kalles *visuell odometri*, og går ut på å identifisere gjenkjennelige former i en sekvens av bilder. Deretter kan piksel-differansen mellom de kjente formene finnes og resultatet brukes til å estimere endring i avstand og rotasjon i 3 dimensjoner.

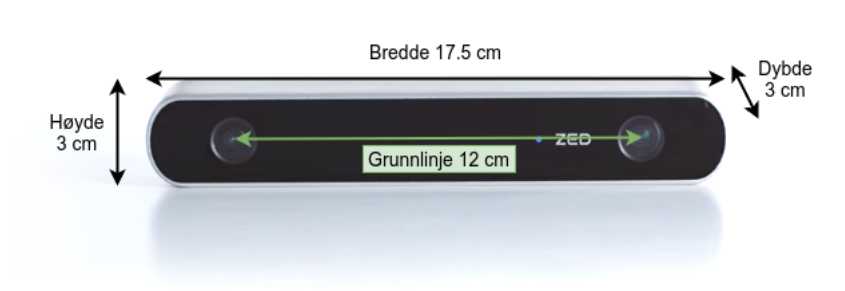
- Sensorens benyttede bidrag til lokalisering

- xy_w : Robotens relative translasjon langs x- og y-akse, forankret i verdenskoordinater.
- θ_z Robotens rotasjon om z-aksen.

2.2.2 Sensor for maskinsyn

Det finnes mange typer sensorer som brukes for å gi roboter syn. Felles for dem er at de forsyner systemet med data som beskriver avstand til objekter innen sensorens synsfelt, i nær-sanntid. Et populært eksempel er laserbaserte sensorer som kan være kostbare. Et godt og ofte rimeligere alternativ er å utnytte den unike geometrien som oppstår ved å benytte to kamera i samme konfigurasjon som mennesker og dyrs øyne; stereosyn.

Stereokamera

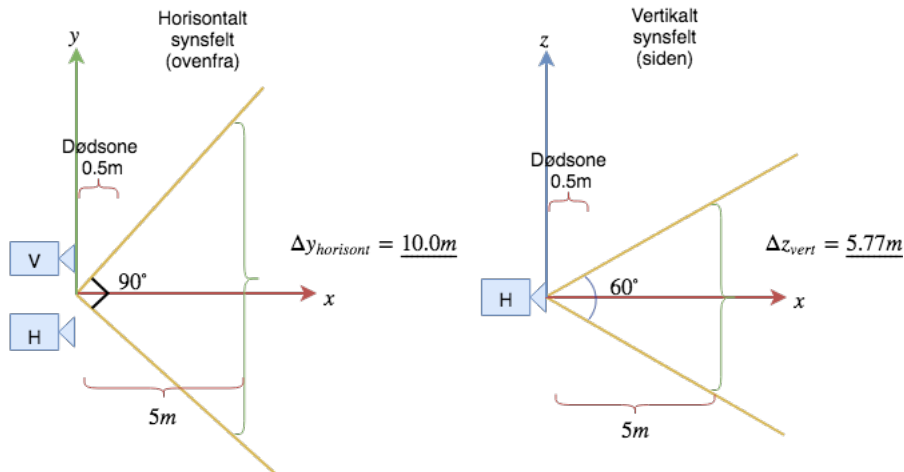


Figur 10: ZED stereokamera [9]

Synsfelt	Horisontal	90°	
	Vertikal	60°	
Rekkevidde	Minimum	0.5m	
	Maksimum	20m	
Bildesensor	Oppløsning	Maksimal	4416x1242 piksler
		Benyttet	2560x720 piksler
	Bildefrekvens	Maksimal	100 bilder/s
		Benyttet	30 bilder/s
Grensesnitt	Dataoverføring	USB 3.0	
	Strømforsyning	5V / 380mA / via USB	
Vekt			159g
Annen funksjonalitet			Visuell odometri, 3 dimensjoner

Tabell 3: Nøkkeldata ZED stereokamera

Stereokameraet benytter to linser med tilhørende bildesensorer horisontalt separert med en *grunnlinje* på 12 cm. Med et synsfelt på hhv. 90° og 60° i horisontal og vertikal retning kan det teoretisk sett oppdage objekter i et felt illustrert i figur 11.



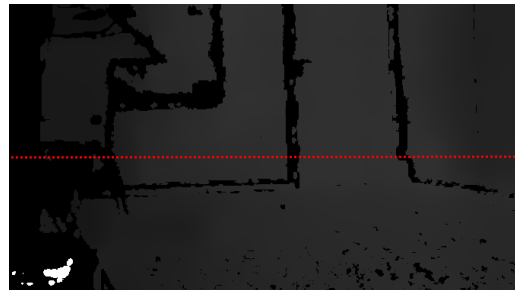
Figur 11: Figuren illustrerer observerbart område ved en objektavstand på 5m i horisontalt og vertikalt plan

Dybdebilder kalkuleres på bakgrunn av et stereobilde-par ved hjelp av en algoritme for stereosyn, som beskrives under:

1. Pre-prosessering av bilder
 - Bilder fra kamera blir kompensert for linsedistorsjon og rektifisert slik at *pikselrader* i bildene får tilsvarende nummerering.
2. Finn korresponderende punkter
 - For å kunne beregne avstand til et (alle) punkt i scenen, må punktet først identifiseres i begge bilder. Slike punkter kalles *korresponderende punkter*. Det er i utgangspunktet en beregningsmessig svært krevende oppgave, men det finnes gode optimaliserte algoritmer som utnytter forenklinger basert på geometrien som oppstår ved et slikt oppsett.
3. Beregn disparitets-kart og rekonstruksjon ved triangulering
 - Disparitet er differansen i pikselkoordinater for de korresponderende punktene, og representerer informasjon om den tredje dimensjonen: dybde. Et disparitets-kart består av dispariteten til alle punkter i bildet. Til slutt brukes punktenes koordinater sammen med dispariteten for å rekonstruere scenen ved triangulering.



(a) Dybdebilde fra stereokamera konvertert fra 32 bit/piksel til 8 bit/piksel gråskala-bilde.



(b) Det samme dybdebildet med linje som illustrerer benyttet informasjon

Figur 12: Dybdebilder

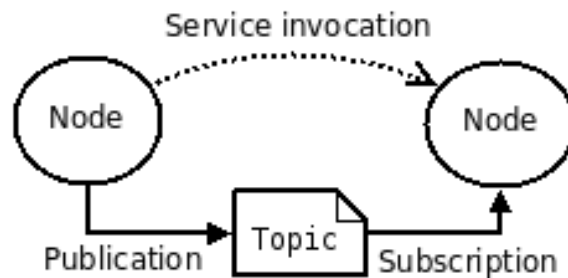
Oppsettet brukt i denne oppgaven benytter ikke all informasjonen som er tilgjengelig i dybdebildet, men henter istedet ut et horisontalt felt med dybdeverdier. Det gjøres fordi det ikke er nødvendig for ruteplanlegger å ha informasjon om hele scenen, den trenger bare vite hva som er foran den i en viss høyde. Beregningsmessig er det også en stor fordel å kutte ned på informasjonen som benyttes i etterkant, men det presiseres at hele dybdebildet blir kalkulert selv om kun en liten del benyttes videre.

2.3 Robot Operating System (ROS)

ROS er ikke et tradisjonelt operativsystem, men en samling av verktøy og biblioteker rettet mot utvikling av roboter, som også inneholder operativsystem-funksjonalitet som lavnivå kontrollsystemer, maskinvareabstraksjon og pakkehåndtering. Det er et åpen kildekode-prosjekt som har sin opprinnelse ved *Stanford Artificial Intelligence Laboratory* i 2007 og som siden 2013 vedlikeholdes av en egen organisasjon kalt *Open Source Robotics Foundation*. ROS er ikke laget for en spesifikk robot-type, men består av en rekke generelle komponenter som kan tilpasses de aller fleste former for roboter. Samlingen inneholder verktøy og biblioteker for blant annet maskinsyn, selvstendig navigasjon, visualisering og datalogging.

ROS installeres ikke som hovedoperativsystem på en datamaskin, men er avhengig av et **Unix**-basert system for å fungere. Operativsystemet **Ubuntu**, som er basert på **Linux**, har størst utbredelse for ROS og er benyttet i denne oppgaven.

2.3.1 Oppbygging



Figur 13: Figur som viser kommunikasjon mellom noder i ROS. [11]

Noder

ROS består av mange små programmer kalt *noder*. Et ferdig konfigurert system for en robot vil inneholde mange noder, hvor hver node kontrollerer hvert sitt undersystem som kamera, motorkontroller, odometri eller visualisering. En node kan lages ved å benytte et *klient*-bibliotek som implementerer ROS-komponenter i flere programmeringsspråk. De mest brukte språkene er **C++** (`roscpp`) og **Python** (`rospy`), og begge benyttes i denne oppgaven.

En node kan enten startes direkte via *kommandolinje* eller via et egendefinert *skript* kalt `launch`-filer, som benytter skriptspråket **XML**. En node har ofte mange *parametre* som må settes for å tilpasse til brukerens spesifikke oppsett. Parametre kan spesifiseres i `launch`-filer, men ved et viss antall blir det tungvint og da benyttes egne konfigurasjonsfiler som struktureres med skriptspråket **YAML**. [12]

Messages

Noder kan koble seg opp mot hverandre og kommunisere ved hjelp av forhåndsdefinerte meldinger. Det finnes f.eks en meldingstype for posisjonsdata og en annen for strømming av bilder. [13]

Topic

Noder kan kommunisere direkte med hverandre, men finner hverandre først via et `topic`. En node som produserer en form for data kan *publisere* den til et `topic`, mens en node som bruker data kan *abonnere* på et `topic`. Et enkelt `topic` kan ha flere samtidige *abonnenter* og *utgivere*, og en enkelt node kan publisere og abonnere på et eller flere `topic`. Et typisk system inneholder mange `topic`, og de navngis på grunnlag av meldingene som publiseres der. Eksempelvis kan et `topic` kalt *posisjon* publiseres til fra flere noder som produserer lokaliseringsdata (motorodometri, kamera), mens det gjerne bare er én node som tar imot og sammenfatter data fra sensorene. [14]

Services

En `service` eller en tjeneste er den andre metoden noder kan bruke til kommunikasjon. En node kan tilby en tjeneste for informasjonsutveksling når det er hensiktsmessig at en node får bekreftelse på mottatt informasjon, i motsetning til en kontinuerlig datastrøm hvor det ikke er nødvendig eller hensiktsmessig med bekreftelse for hver melding. En tjeneste tilbys typisk av en node for å sette eller hente parametre mens den eksekverer. [15]

Master

ROS `master` fungerer som en *navnetjener* for noder. Når en node publiserer til et `topic`, vil den ikke starte sending av data før `master` har gitt beskjed om at det finnes en abonnent. `Master` forteller også nodene om hverandre slik at overføring kan begynne. Et nettverk som sender data direkte mellom enheter og som reguleres av en slik tjener kalles et *peer-to-peer* nettverk, og det er nettopp dét nettverket av ROS-noder er. `Master` holder også kontroll på hvilke noder som abonnerer/publiserer på de ulike topics. [16]

Bag

`Bag` er et format for lagring av data i ROS. Verktøyet kalt `roscpp` abonnerer på valgte `topic` og lagrer all data til harddisk. Det samme verktøyet kan også spille av bag-filen i etterkant, noe som er svært nyttig for analyse av systemet under testing og utvikling. Under avspilling vil innhentet data publiseres på opprinnelige `topic` og dermed kan analyseverktøy benyttes som om roboten virkelig utfører oppgaver. [17]

ROS Graph

Konseptene som er beskrevet ovenfor utgjør tilsammen **computation graph**; et **peer-to-peer** nettverk av prosesser (noder) som produserer data i ulike former (meldinger) til **graph**. En slik organisering av prosesser gjør at noder kan kommunisere med hverandre uten å være avhengig av å kjøre på samme fysiske enhet. Det utgjør en av styrkene til ROS; et desentralisert nettverk av prosesser kan samhandle på tvers av fysiske enheter og fungere sammen som ett.

ROS console

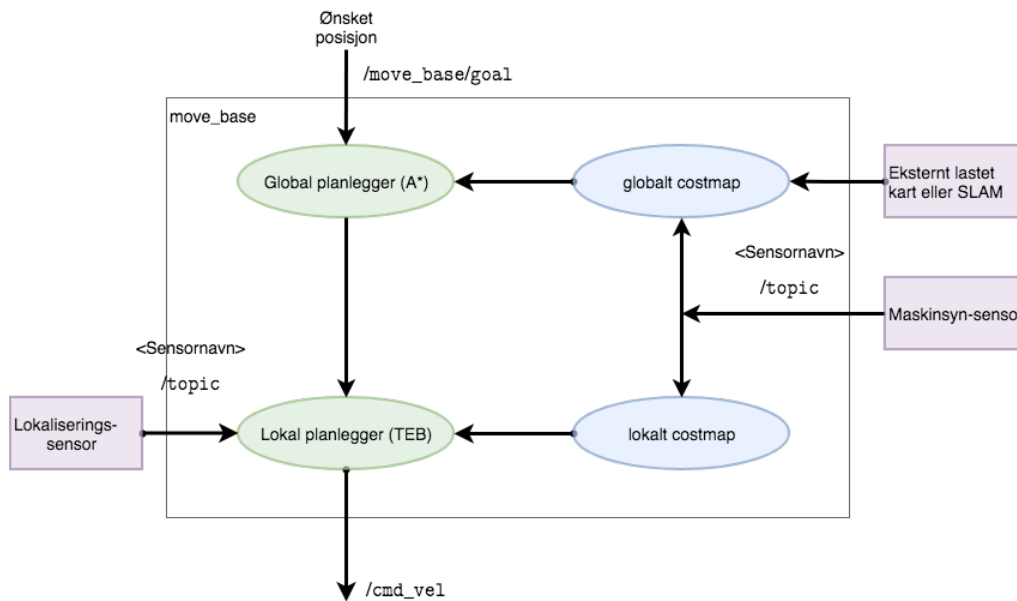
ROS console er der alle noder sender informasjons- og feilmeldinger. Det finnes mange nivå av meldinger, hvor det dypeste beskriver detaljert hva en node foretar seg og brukes kun til feilsøking. Det høyeste nivået består av informasjonsmeldinger til bruker, som eksempelvis inneholder informasjon om oppstart og nedkjøring av noder.

Andre verktøy

Rviz (ROS visualization) (*norsk: ROS visualisering*) er et verktøy som visualiserer robotens *virtuelle bilde* av verden. Den inneholder en modell av roboten samt **costmap**, som introduseres i neste delkapittel. Det kan også benyttes til å sende enkle *kommandoer* til Spurv, i form av koordinater som beskriver ønsket posisjon for roboten. Alle skjermbilder i rapporten kommer fra dette verktøyet.

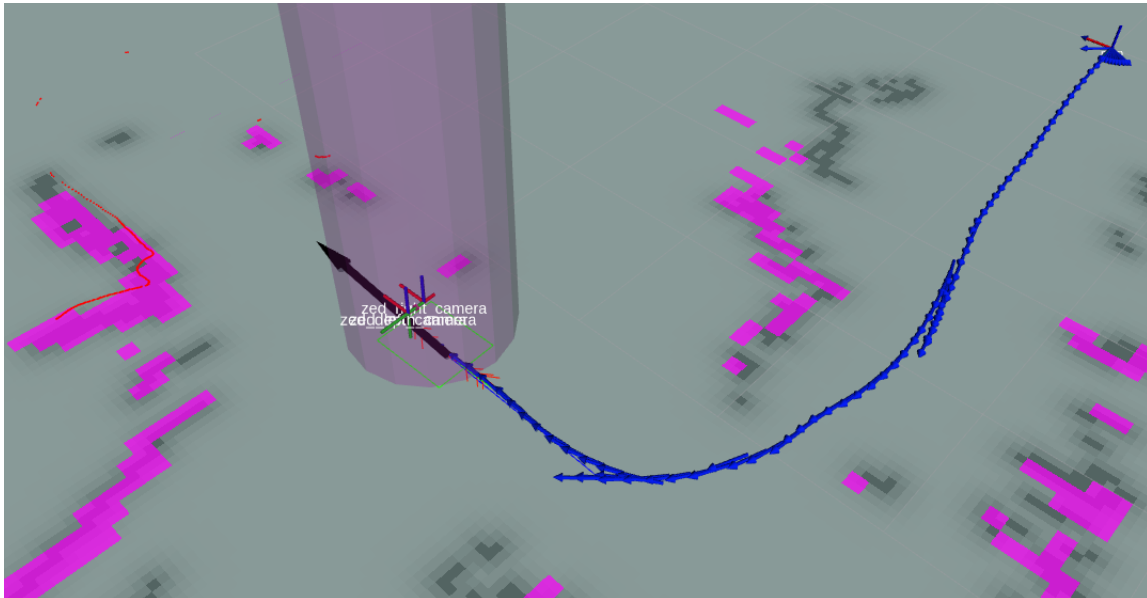
2.3.2 Selvstendig navigasjon

For at en robot skal kunne navigere på egenhånd er det en del systemer som må være på plass. Ulike sensorer henter inn data fra den fysiske verden, som danner et beslutningsgrunnlag for hvordan roboten skal oppføre seg i en gitt situasjon. I ROS finnes en samling av pakker, kalt **move_base**[10] (*norsk: flytt plattform*) som tar inn data fra sensorer, behandler dem, og gir ut hastighetskommandoer som er basert på nåværende posisjon, målposisjon og informasjon om hindringer i nærområdet. For at denne samlingen av pakker skal fungere som ønsket er det nødvendig å sette seg inn i hvordan den er bygget opp, for deretter å være i stand til å konfigurere det spesifikke systemet benyttet i denne oppgaven.



Figur 14: Oversikt `move_base` som viser sammenhengen mellom de ulike `costmap` og ruteplanleggere. Når det mottas en ønsket posisjon starter global planlegger med å hurtig lage en rute basert på data fra globalt `costmap`. Deretter sendes ruten til lokal planlegger som benytter den til å lage hastighetskommandoer, som sendes til topic `command_velocity` (`cmd_vel`) for videre konvertering til motorkommando. Det kommer også frem at maskinsynsensor er grunnlaget for både lokalt og globalt `costmap`.

Costmap



Figur 15: Figuren viser roboten representert i rviz med både lokalt og globalt *costmap* samt de ulike lagene representert med fargekoder. Hindringer har fargen lilla i lokalt *costmap* og grå for globalt *costmap*. Skillet mellom lokalt og globalt går der hvor lilla hindringer slutter.

Et *costmap* er et kart som inneholder informasjon om kjente objekter i xy-planet. Det er organisert som en matrise hvor hvert element angir sannsynligheten for at en *celle* er okkupert av et objekt. En celles verdi kan variere mellom 0 (tomt rom) og 255 (sikker hindring). Idet roboten starter opp er *costmap* fylt med celler med ukjent status, og den vil straks sette igang med å oppdatere verdiene basert på egen posisjon og informasjon om nærområdet fra en maskinsyn-sensor. Etterhvert som roboten beveger seg legges informasjon om observerte hindringer og tomt rom inn.

Selv om en celle kan ha en verdi mellom 0 og 255 er det kun tre verdier som brukes når ROS skal finne en rute til et gitt mål. Celleverdiene terskles basert på brukersatte verdier til 3 kategorier: `LETHAL_OBSTACLE` (*farlig område*), `NO_INFORMATION` (*ikke utforsket/ukjent område*) og `FREE_SPACE` (*tomt område*).

ROS benytter to typer *costmap* parallelt: lokalt og globalt *costmap*. De er stablet oppå hverandre og begge benyttes kontinuerlig.

Det globale representerer hele robotens verdensbilde og inneholder således informasjon om store områder, og beregnes derfor mindre hyppig enn det lokale. Størrelsen er en avveining mellom tilgjengelig kapasitet og nødvendig størrelse for aktuelt bruksområde, og bestemmes av bruker. Det globale *costmaps* hensikt er å ta vare på informasjon om topografi i besøkte områder slik at roboten enkelt kan planlegge effektive ruter ved neste besøk. Det kan også inneholde informasjon fra et eksternt

kart dersom det finnes og lastes inn.

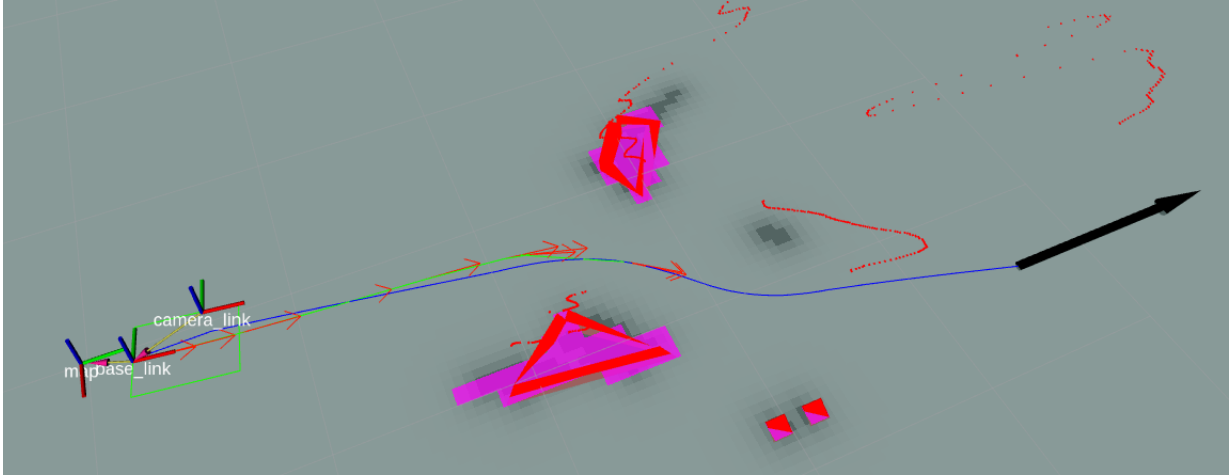
Det lokale dekker et mindre område (typisk 5 x 5 m) men oppdateres hyppigere enn det globale. Det følger alltid roboten og består av data fra visuelle sensorer som gjør at det kan sies å inneholde informasjon om omgivelsene i (nær) sanntid. Hensikten med lokalt `costmap` er å representere informasjon om umiddelbare omgivelser slik at systemet får hurtig oppdatert informasjon og dermed kan unngå kollisjon. Når objekter kommer utenfor sensors rekkevidde og dermed forsvinner fra lokalt `costmap`, tar det globale over informasjonen.

Hvert `costmap` består av flere *lag* som inneholder en bestemt type informasjon.

- **Static map layer** (*statisk kartlag*): Det statiske kartlaget inneholder informasjon som på forhånd er kjent om et område, og kan kalles et tradisjonelt kart. Slik data endres sjelden.
- **Obstacle map layer** (*hindringslag*): Hindringslaget består av data fra visuelle sensorer, som beskrevet over. Dataen beholdes når sensoren er utenfor rekkevidde, slik at det bygges opp et slags kart.
- **Inflation layer** (*utvidelseslag*): Et lag som legger til en sikkerhetssone rundt celler som er definert som `LETHAL_OBSTACLE`, slik at jo kortere distansen til et objekt er assosieres med en tilsvarende høy kost. Denne kosten beregnes utfra en *kostfunksjon*, og dens negative stigningsgrad bestemmes via parameteren `cost_scaling_factor`. Kurvens totale lengde avgrenses av parameteren `inflation_radius`. Parametrene bestemmer sammen hvor stor *risiko* systemet skal forbinde med en hindring, og dermed hvilke avstander som er trygge.

Kriteriet for *når* en sensor skal plassere `LETHAL_OBSTACLE` og `FREE_SPACE` i lokalt `costmap` er basert på parametrene `obstacle_range` og `raytrace_range`. Hvis målt distanse til en hindring er mindre enn `obstacle_range` defineres det som en hindring, mens hvis distansen til hindring er mindre enn `raytrace_range` merkes alt rom mellom robot og hindring som tomt. [29]

Ruteplanleggingsalgoritmer



Figur 16: Figur som viser grafisk representasjon av lokal (grønn linje med røde piler) og global (blå linje) rute

En planleggingsalgoritme har som oppgave å finne kjørbare ruter fra A til B, for deretter evaluere dem og velge ut den beste ruten. ROS brukes to slike algoritmer samtidig; *lokal* og *global* planlegger. For hver av disse finnes det flere alternativer, mens denne oppgaven benytter **Timed Elastic Bands**[30] som lokal planlegger og **A*** (A star)[31] som global planlegger.

Den globale planleggeren leter etter en optimal rute fra startposisjon til målposisjon med (kortest mulig) distanse som kriterium. Grunnlaget for å lage denne ruten er basert på det globale **costmap** og resulterer i en rute bestående av en sekvens av veipunkt. Global planlegger beregner sin rute kun én gang per ønsket målposisjon.

Lokal planlegger benytter seg deretter av veipunktene og forsøker å finne den beste hastigheten ($\dot{x}_r, \dot{\theta}_r$) for å nå neste veipunkt ved å simulere en kort periode fram i tid (4 m frem i tid, justerbar) ved ulike hastigheter. Potensielle hastighetsprofilers øvre begrensning bestemmes av parametre, mens faktisk hastighet i realiteten bestemmes av algoritmens oppfattelse av topografien. Den beste hastighetsprofilen blir så utført og prosessen starter på nytt. Lokal planlegger beregner kontinuerlig den beste ruten basert på observasjoner underveis.

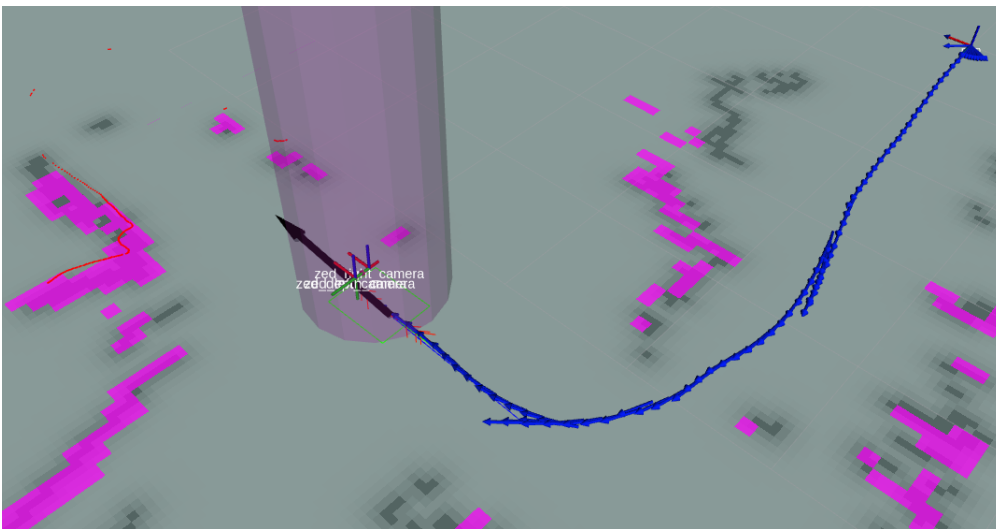
Timed Elastic Bands

Timed Elastic Bands (TEB) bearbeider veipunkt produsert av en global planleggingsalgoritme istedenfor å produsere dem selv. Dette er å foretrekke siden en robot ofte opererer i et miljø den ikke har fullstendig oversikt over, og hvor det finnes dynamiske hindre som mennesker eller andre roboter. Det vil derfor lønne seg å gjøre den initielle planlegging hurtig, for deretter å ta hensyn til detaljer. TEBs grunnidé er å strekke og forskyve eksisterende ruter, på samme måte som man kan strekke en

elastisk strikk (elastic band), og på den måten unngå å bruke unødvendig prosesseringskraft. Algoritmen tar også hensyn til tid, slik at den har mulighet til å finne en optimal rute basert på tid istedenfor avstand, som ikke nødvendigvis alltid er tilsvarende.

TEB er benyttet i denne oppgaven fordi den er tilpasset bruk på roboter som benytter Ackermann-styring.

2.3.3 Referansekoordinatsystem



Figur 17: Figur som viser benyttede referansekoordinatsystem. Øverst til høyre er odom og map (som betyr at ingen feil er detektert fra relative sensorer), mens `base_link` er forankret i robotens modell, hvor også sensorenes koordinatsystem befinner seg.

Et system som skal ta seg av navigasjon for en robot er avhengig av å ha klart definerte koordinatsystem å forholde seg til. Det er viktig for brukeren at det inneholder klare og entydige definisjoner som samtidig muliggjør komplekse konfigurasjoner.

I styringsdokumentet *REP 105: Coordinate Frames for Mobile Platforms* [5] er det definert standardiserte navn for ulike relevante referansekoordinatsystem med tilhørende spesifiserte bruksområder.

base_link

`base_link` (norsk: *grunnledd*) er forankret i robotens posisjon. Nøyaktig plassering på roboten varierer utifra hvilken kinematisk modell roboten benytter, og er her beskrevet nærmere i neste kapittel. Plassering er midt på akslingen mellom robotens bakhjul.

odom

odom (forkortelse for *odometry*, norsk: *det å holde kontroll på tilbakelagt distanse*) er et referansekoordinatsystem som definerer et fast utgangspunkt eller referanse for robotens translasjon og rotasjon målt med *relative* sensorer. Siden usikkerheten ved denne formen for posisjonsestimering stiger i takt med tid siden oppstart og tilbakelagt distanse, er **odom** alene en dårlig referanse over lengre tidsrom. Fordelen med slike målinger er at de alltid vil være *kontinuerlig* i motsetning til f.eks en GPS-måling som er diskret av natur.

base_link forflytter seg i forhold til **odom** når hastighet registreres fra relative sensorer. Slike posisjonsestimat kalles *relative* fordi de beskrives *relativt* til en posisjon ved et tidligere tidspunkt.

map

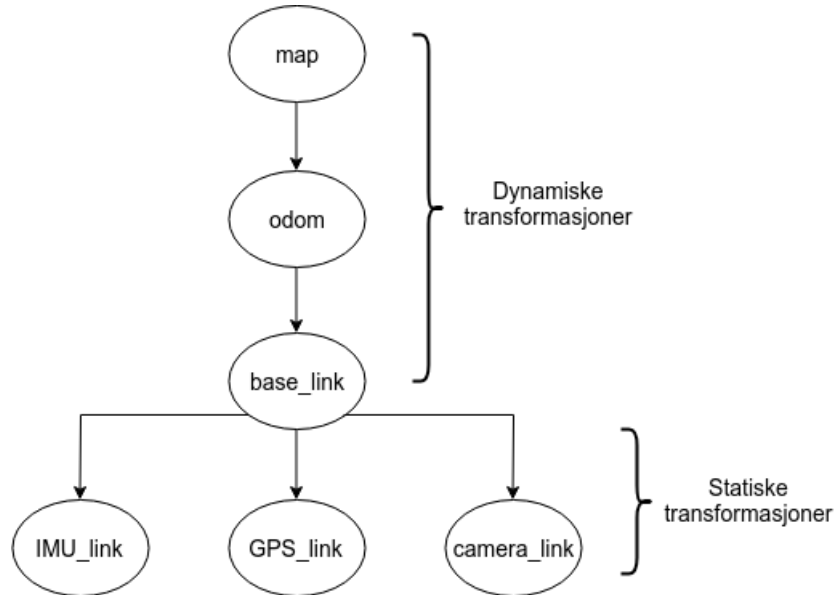
map (norsk: *kart*) er på samme måte som **odom** en fast referanse. Robotens posisjon i **map** baseres på *diskret* data, som medfører at posisjonen vil endre seg i form av *diskrete hopp* ved hver oppdatering fra sensor. **base_link** sin translasjon og rotasjon i forhold til **map** vil være en god referanse over lengre tidsperioder siden posisjonen kontinuerlig beregnes fra nye sensormålinger, men vil derimot ikke egne seg til produsere posisjonsestimat som kan brukes ved f.eks kartlegging av et område med maskinsyn.

utm

utm er et referansekoordinatsystem som benyttes ved transformasjon av globale koordinater til lokale, og er definert spesielt for denne oppgaven. En grundigere forklaring finnes i kapitlet 2.5 "Globale koordinatsystem".

En utbredt posisjons-kilde for **map** er en GNSS-mottaker. Posisjonen den estimerer kalles *global* fordi den kommer relativt til et eksisterende (og globalt) koordinatsystem, her **World Geodetic System**, som også er beskrevet nærmere i kapittel 2.5.

Forholdet mellom referansekoordinatsystem



Figur 18: Et typisk transformasjonstre

Hvert referansekoordinatsystem har fordeler og ulemper, men når de kombineres faller fordelene sammen og ulempene forsvinner i stor grad. Sammenhengen er modellert som en trestruktur, hvor hvert enkelt referansesystem er begrenset til å ha én enkelt *forelder* men et ubegrenset antall *barn*. Med bakgrunn i forklaringen over er det naturlig å tenke seg at `base_link` burde ha både `odom` og `map` som foreldre, men grunnet denne restriksjonen må det være som vist i figur 18.

Referansekoordinatsystemenes posisjon i rommet endres ved at en node produserer og publiserer en *dynamisk* transformasjon mellom to systemer. Dette betyr at dersom roboten, her representert ved `base_link`, beveger seg, vil ny data fra en relativ sensor bidra til en endring i transformasjonen `odom`→`base_link`. Tilsvarende vil en posisjonsoppdatering fra en GNSS-mottaker endre transformasjonen `map`→`base_link` ved å først motta transformasjonen `odom`→`base_link` for så å bruke den informasjonen til å publisere `map`→`odom` som resulterer i transformasjonen `map`→`base_link`.

Verdien for transformasjonen `map`→`odom` tilsvarer derfor *feilen* i transformasjonen `odom`→`base_link` (relative sensorers feil). `map`→`odom` er derfor det globale posisjons-estimatets forsøk på å kompensere for relative sensorers feil.

Statiske transformasjoner brukes for å beskrive fastmonterte enheters rotasjon og translasjon i forhold til `base_link`. En slik transformasjon vil aldri endre seg, men er høyst nødvendig å definere korrekt ved montering av nytt utstyr. Eksempelvis vil et kamera som er definert med feil rotasjon kunne rapportere kommende hindringer bak roboten istedenfor foran.

2.3.4 Konvensjoner for koordinatsystem og enheter

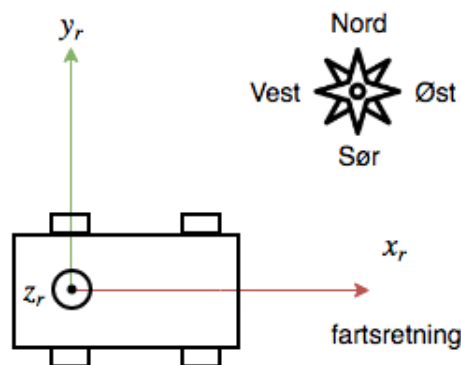
I ROS er konvensjoner for måleenheter og koordinatsystemer definert i styringsdokumentet *REP 103: Standard Units of Measure and Coordinate Conventions* [4]. Her kommer det frem følgende retningslinjer:

Måleenheter

Fysisk størrelse	Enhet
Lengde	meter [m]
Tid	sekund [s]
Strømstyrke	ampere [A]
Vinkel	radian [rad]
Magnetisme	tesla [T]
Temperatur	celsius [°C]

Koordinatsystem

- Orientering, akser i forhold til fartsretning
 - x_r : fram
 - y_r : venstre
 - z_r : opp
- Orientering ved kartesisk representasjon av geografiske lokasjoner følger standarden kalt *ENU (East North Up)*
 - x_r : øst
 - y_r : nord
 - z_r : opp



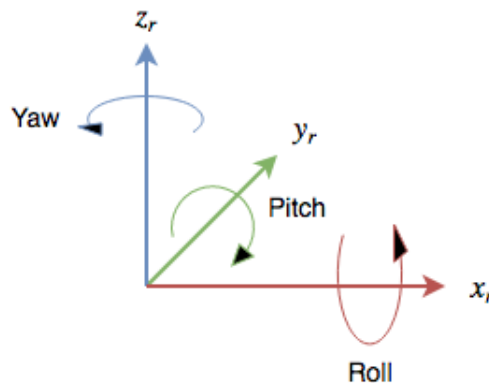
Figur 19: Koordinatsystemet *base_link* for *Spurv*, sett ovenfra. z_r peker opp/ut av bildet.

- Representasjon av rotasjon, prioritert rekkefølge:
 1. Kvaternioner
 2. Rotasjonsmatrise
 3. Roll, Pitch, Yaw (RPY) om XYZ
 4. Eulervinkler, yaw, pitch, roll om ZYX

Alle koordinatsystem følger høyrehåndsregelen, som blant annet innebærer at når et legeme roteres *mot klokken* skal rotasjon om z -aksen, **yaw**, øke.

Her kommer det fram at rotasjon kan representeres på forskjellige måter, og for oppgaven brukes både kvaternioner og RPY.

Roll, pitch og yaw



Figur 20: Figuren viser hvordan rotasjon om akser beskrives med **Roll**, **Pitch** og **Yaw**.

Betegnelsen kommer opprinnelig fra flybransjen, men brukes også for andre farkoster. Systemet navngir rotasjon om de ulike aksene i et tredimensjonalt koordinatsystem som følger høyrehåndsregelen, slik at det er enkelt å referere til dem.

Kvaternioner

Kvaternioner er et tallsystem som utvider de komplekse tallene, og kan blant annet kan brukes til å angi rotasjon i 3 dimensjoner. Systemet er i utbredt bruk innen robotikk av flere grunner:

- Lider ikke av problemet **gimbal lock** som for eulervinkler. Det er et fenomen som oppstår når to av tre *slingrebøyler* (som modellerer eulervinkler) havner i en konfigurasjon hvor de er parallelle. Det gjør at systemet i praksis mister en dimensjon ved en slik konfigurasjon, og er ikke ønskelig.[32]

- Størrelses- og beregningsmessig effektiv i forhold til rotasjonsmatriser. For å beskrive en rotasjon i 3 dimensjoner trenger kvaternioner kun 4 skalarverdier, mens rotasjonmatriser krever 9. I tillegg er multiplikasjon av kvaternioner mer effektivt enn multiplikasjon av matriser.

En rotasjon med kvaternioner beskrives som:

$$x + y\mathbf{i} + z\mathbf{j} + w\mathbf{k} \quad (20)$$

hvor x , y , z og w er reelle tall og \mathbf{i} , \mathbf{j} og \mathbf{k} er de fundamentale enhetskvaternionene. Enhetsrotasjonen, dvs. ingen rotasjon, ser ut som følgende:

$$0 + 0\mathbf{i} + 0\mathbf{j} + 1\mathbf{k} \quad (21)$$

[27]

2.4 Bakgrunn for sentrale noder lagt til i ROS

I dette delkapitlet beskrives de mest essensielle nodene som er tatt i bruk i løsningen for denne oppgaven.

2.4.1 Sensorfusjon

- ROS-pakke: `robot_localization` [25]

Sensorfusjon går ut på å kombinere posisjonsdata fra flere kilder for å produsere et posisjonsestimat som er bedre enn hva som kan oppnås med én enkelt kilde. Denne prosessen kalles også *tilstandsestimering*. Ved kombinasjon av posisjonsdata på denne måten kan også data fra sensorer med ulik *samplingrate* benyttes sammen.

Pakken `robot_localization` inneholder tre noder:

- Noder for tilstandsestimering
 - `ekf_localization_node`: Implementasjon av *Extended Kalman Filter (EKF)* (norsk: *utvidet kalman filter*)
 - `ukf_localization_node`: Implementasjon av *Unscented Kalman Filter (UKF)*
- Node for preprosessering av data
 - `navsat_transform_node`: Node for transformasjon av *sfæriske* koordinater til *kartesiske* koordinater

Kalman-filter

Som navnet på nodene avslører er tilstandsestimering implementert ved bruk av Kalman-filter. *EKF* er den ulineære varianten av Kalman-filter som betyr at det kan brukes på ulineære systemer ved å linearisere modellen rundt et arbeidspunkt. Introduksjon av en modell har den fordelen at data med ulik *samplingrate* kan benyttes sammen for å produsere et estimat.

I praksis er forskjellen på de to nodene for tilstandsestimering at *UKF* er beregningsmessig mer krevende enn *EKF*, men kan gi bedre resultater. Ved utprøving er det funnet at *EKF* gir gode nok resultater og er derfor benyttet videre.

Et Kalman-filter er en estimator som på bakgrunn av en systemmodell og fysiske målinger med tilhørende usikkerhet kan estimere kjøretøyets posisjon, eller *tilstand*, mer nøyaktig enn det som kan finnes fra en enkelt måling alene. Det er spesielt nyttig ved tilfeller der målinger inneholder spredning i data, som fra en GNSS-mottaker.

\mathbf{x} er tilstandsvektoren som beskriver systemets posisjon, hastighet, aksellerasjon, rotasjon og rotasjons hastighet, som tilsammen gir 15 tilstandsvariabler da de beskrives i 3 dimensjoner.

For å estimere \mathbf{x}_k basert på forrige tidssteg brukes en kinematisk modell, f :

$$\hat{\mathbf{x}}_k = f(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1} \quad (22)$$

hvor \mathbf{w} kalles *prosessens støy* og representerer estimatets usikkerhet. Usikkerhetens varians beskrives med en *kovariansmatrise* kalt prosesskovarians \mathbf{Q}_k .

Den fysiske målingen, \mathbf{z} , fra en sensor beskrives ved hjelp av målemodellen h :

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k \quad (23)$$

hvor \mathbf{v} kalles *målestøy* og representerer målingens usikkerhet. Usikkerhetens varians beskrives med kovariansmatrisen kalt målekovarians, \mathbf{R}_k .

Kalman-filteret implementeres med en algoritme som består av to hoveddeler:

1. **Prediksjon:** Tilstanden ved nåværende tidssteg predikeres basert på modell. Det resulterer i *a priori* predikert tilstandsestimat, $\hat{\mathbf{x}}_{k|k-1}$, og predikert estimat-kovarians, $\mathbf{P}_{k|k-1}$, som tilsvarer variansen i estimatets usikkerhet før fysisk måling.
2. **Oppdatering:** Fysisk måling ved nåværende tidssteg tilføres resultatet i steg 1. *a posteriori* predikert tilstandsestimat $\hat{\mathbf{x}}_{k|k}$ og predikert estimat-kovarians $\mathbf{P}_{k|k}$ representerer tilsammen det beste mulige posisjonsestimaten inkludert usikkerhet.

Resultatet fra steg 2 sendes så tilbake til steg 1 for å estimere tilstanden for neste tidssteg, og denne syklusen går kontinuerlig med en frekvens på 30 Hz for Spurv.

En beregningsmessig fordel med Kalman-filter er at det kun krever *nåværende måling* og *forrige estimat* for å beregne *neste estimat*, i motsetning til andre algoritmer som krever lang historikk for å beregne neste verdi. Det gjør at det er mulig å implementere slik estimering på mobile enheter som har begrensede ressurser.

Publisert informasjon:

- Transformasjon
 - `map` → `odom`
 - `odom` → `base_link`
- Annet
 - `/odometry_filtered`: Robotens posisjon og rotasjon i referansekoordinatsystemet `odom` basert på relative sensorer. (`nav_msgs/Odometry`)
 - `/odometry_filtered_map`: Robotens posisjon og rotasjon i referansekoordinatsystemet `map` basert på relative og globale sensorer. (`nav_msgs/Odometry`)

Transformasjon av global posisjoneringsdata

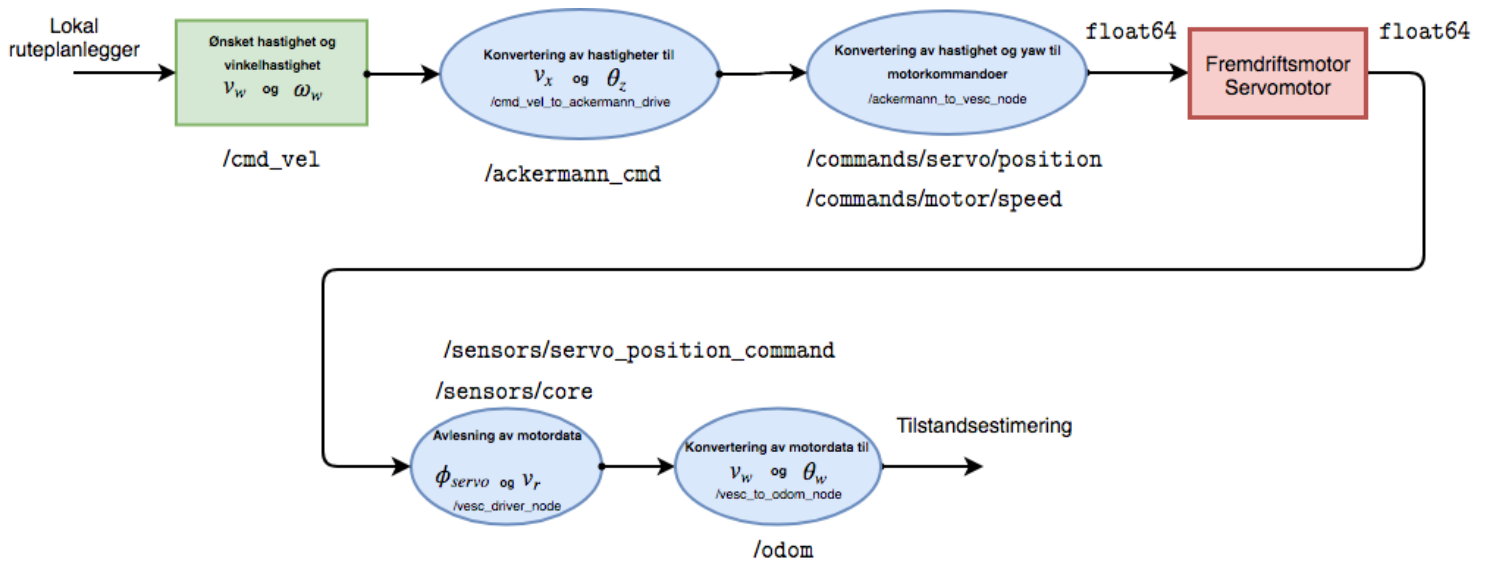
Noden `navsat_transform_node` transformerer data kontinuerlig fra GNSS-mottaker i form av lengde-og breddegrader, via UTM-koordinater, til robotens koordinatsystem `map`. En grundig forklaring av koordinatsystem og konvertering mellom dem finnes i kapittel 2.5.2.

Publisert informasjon:

- Transformasjon
 - `utm` \rightarrow `map`
- Referansekoordinatsystem
 - `utm`

2.4.2 Hastighetskontroll

Figur 21 viser en oversikt over noder som er involvert i å konvertere ønsket hastighet til en datatype som er lesbar for motorer, samt lese av reell hastighet og gjøre dem om til data som deretter brukes til å estimere posisjon.



Figur 21: Figuren viser noder som setter ønsket hastighet og konverterer dem tilbake til posisjon.

2.5 Globale koordinatsystem

For å representere en vilkårlig posisjon på jordoverflaten, benyttes et koordinatsystem sammen med en modell som representerer jordens elliptiske form. Delkapitlet presenterer og forklarer konsepter og begreper innen som benyttes videre.

2.5.1 WGS 84

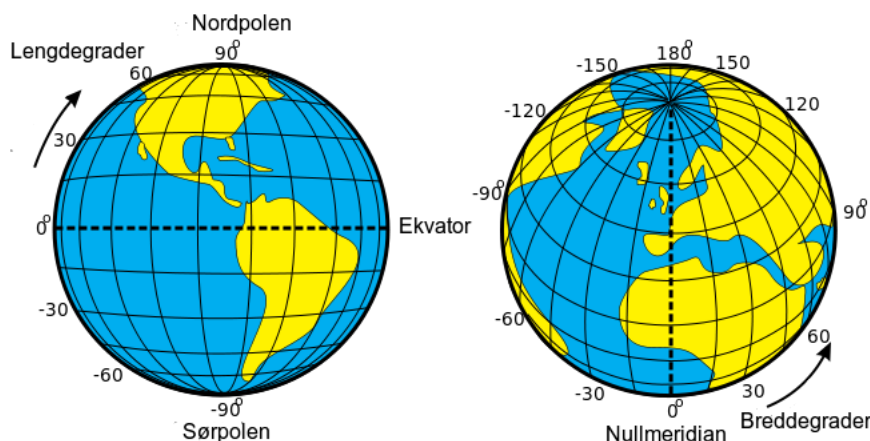
WGS 84 (*World Geodetic System 1984*) er en utbredt standard som benyttes for å beskrive lokasjoner på jorden. Systemet består av en *referanse-ellipsoide* som beskriver jordens form matematisk, og kalles også *datum*, som betyr utgangspunkt eller nullpunkt. I tillegg inneholder standarden et *kartesisk* koordinatsystem som kan beskrive en vilkårlig lokasjon vha. tre koordinater, x y z , med jordens massesenter som nullpunkt. Et slik koordinatsystem er upraktisk for operasjoner på jordens overflate, derfor benyttes andre koordinatsystemer som baseres på *modellen* i WGS 84.

2.5.2 Koordinatsystem

Det finnes mange ulike koordinatsystem som er utformet for forskjellige formål. Avsnittet beskriver relevante koordinatsystem for denne oppgaven.

Geografisk koordinatsystem

Lengde- og bredde-grader er et eksempel på et mye brukt *geografisk* koordinatsystem som deler opp jordens overflate etter linjer parallelle med ekvator (lengdegrader) og linjer parallelle med *nullmeridianen* (breddegrader).



Figur 22: Et geografisk koordinatsystem som lengde- og bredde-grader deler opp jordens overflate med utgangspunkt i ekvator (lengdegrader) og nullmeridianen (breddegrader). [19]

Hensikten med oppdelingen er å etablere et koordinatsystem for å beskrive lokasjoner på jordens overflate. Det er et *sfærisk* koordinatsystem, som medfører at det ikke er *euklidisk*, noe som gjør at distanser i meter ikke kan beregnes direkte. Lengde- og bredde-grader kan heller ikke direkte brukes til å spesifisere lokasjoner i et to-dimensjonalt kart, for det trengs en *projeksjon*. WGS 84 inneholder ikke en projeksjon, derfor brukes ofte projeksjonen UTM, beskrevet i neste avsnitt.

Projisert koordinatsystem

Et *projisert* koordinatsystem transformerer jordens tre-dimensjonale elliptiske form til et to-dimensjonalt plan. UTM (*Universal Transverse Mercator*) er en slik *projeksjon*. Systemet inneholder også et *kartesisk* koordinatsystem som bruker det metriske systemet til å beskrive distanser. Et kartesisk koordinatsystem har den fordelen at det er *euklidisk*, som gjør det enkelt å beregne distanser i meter.

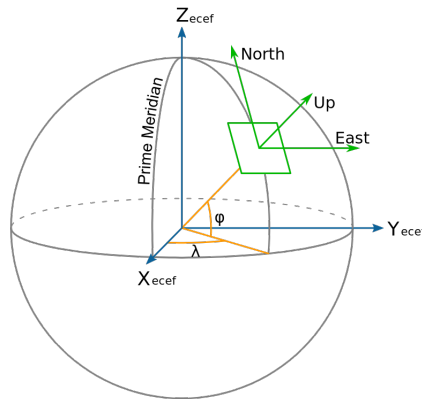
UTM deler jordkloden opp i rektangulære soner som hver har sin egen projeksjon. Hver enkelt sone har derfor et eget nullpunkt og lokasjoner defineres utfra en sones nullpunkt i meter.



Figur 23: Oversikt UTM-soner for Europa [20]

Lokale koordinatsystem

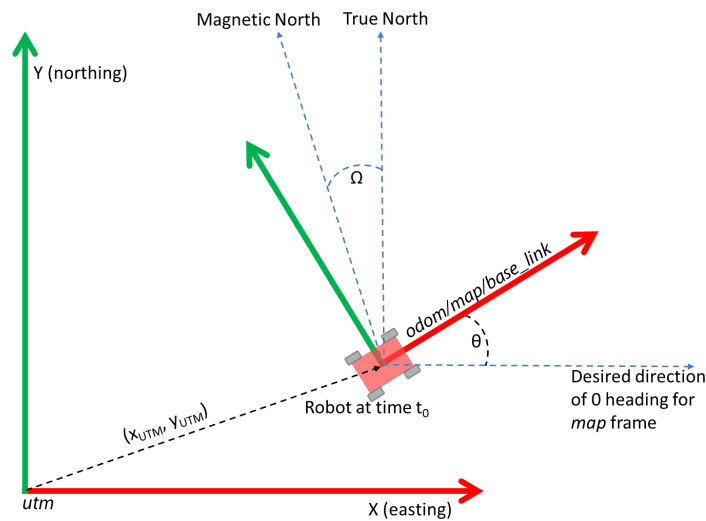
Et *lokalt* koordinatsystem er forankret i fartøyets masse, og er således ikke globalt. To lokale systemer benyttes i denne oppgaven; ENU (*East North Up, norsk: øst nord opp*) og NED (*North East Down, norsk: Nord Øst Ned*). Begge er kartesiske koordinatsystem, og aksene defineres etter høyrehåndsregelen slik at de tilsvarer x, y, z for sine respektive navn.[22]



Figur 24: Figuren viser hvordan koordinatsystemet **ENU** er definert i forhold til et koordinatsystem forankret i jordens senter. **NED** defineres tilsvarende ved å bytte om aksene som i sitt navn.[23]

Konvertering mellom koordinatsystem

Siden både UTM og lengde- og bredde-grader benytter samme datum som referanse, er det enkelt å konvertere mellom dem. For denne oppgaven er det aktuelt å transformere mellom koordinater fra GNSS-sensor til robotens koordinatsystem, slik at global posisjonsdata benytter samme referansekoordinatsystem som andre sensorer.



Figur 25: Figuren illustrerer konseptet for transformasjon mellom koordinater fra GNSS-sensor til robotens koordinatsystem. [24]

Figuren over viser robotens referansekoordinatsystem `map`, `odom`, `base_link` samt et ekstra system kalt `utm` som publiseres av noden `navsat_transform`. Det plasseres i lokasjonen til nåværende UTM-sones nullpunkt.

For å konvertere lengde- og breddegrader til robotens koordinatsystem, `map`, må transformasjonen `utm` \rightarrow `map` beskrives. Da må lengde- og breddegrader først konverteres til UTM-koordinater for å få X_{UTM} og Y_{UTM} . I tillegg må robotens rotasjon

iforhold til øst, θ , hentes fra en sensor som rapporterer rotasjon i forhold til himmelretning og kompenserer for magnetisk deklinasjon, Ω , for aktuell breddegrad.

Så lenge robotens sensorer og koordinatsystem følger ROS-standard kan transformasjonen `utm` \rightarrow `map` beregnes ved T_{utm}^{map} som inneholder X_{UTM}, Y_{UTM} og θ . [?]

3 Konstruksjon av forbedret navigasjonssystem

Konstruksjonskapitlet beskriver hvordan systemet er bygget opp og hvordan det fungerer på overordnet nivå. Her presenteres egenutviklet programvare samt nødvendige endringer som er gjort i kildekode skrevet av andre. I tillegg gjennomgås parametre som er funnet å ha betydelig påvirkning for oppnådd resultat.

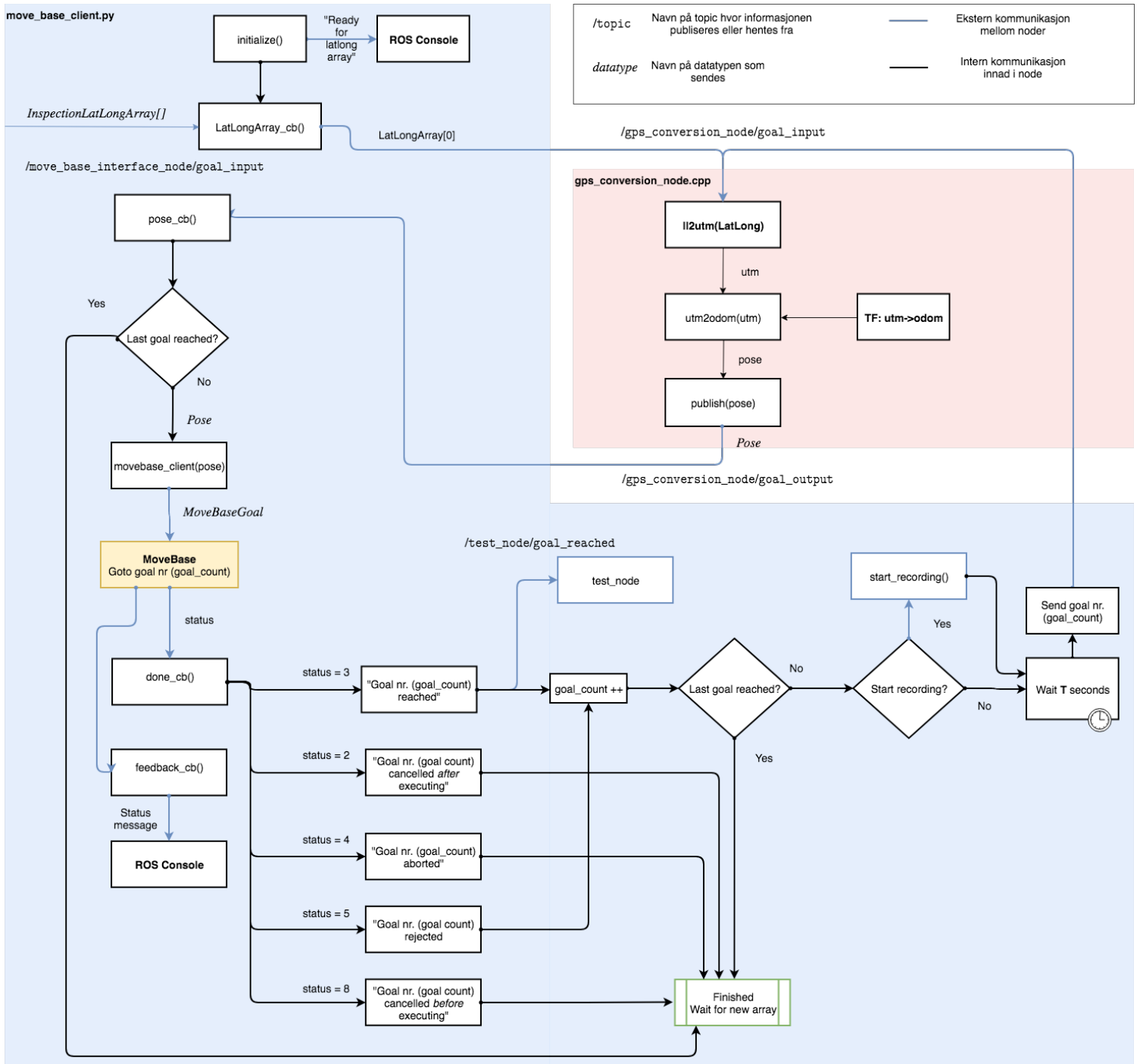
3.1 Systemets oppbygging

3.1.1 Rutefølging med oppgaver

For at roboten skal kunne motta en rute som inneholder tilleggsinformasjon om oppgaver fra operatør, er det utviklet en egen løsning. Ruten skal kunne inneholde koordinater oppgitt i lengde- og breddegrader, slik at programmet må kunne transformere mellom relevante koordinatsystem. Løsningen består av flere noder som kommuniserer sammen, og er utviklet i programmeringsspråkene Python og C++.

Flytskjema

Figur 26 viser en oversikt over programvaren utviklet for rutefølging. Alle funksjoner som er **uthevet** er funksjoner som er laget av andre, resterende er utviklet for denne oppgaven.



Figur 26: Flytskjema, rutefølging

Beskrivelse `move_base_client`

Hovedprogrammet `move_base_client` er utviklet i Python og har som funksjon å ta imot koordinater med tilleggsinformasjon fra bruker, samt ta seg av samhandling med `move_base`, som sørger for å bevege roboten trygt. Det første som skjer når programmet startes er at noden `/move_base_interface_node` startes og registreres som en ROS-node, samt at publiserings- og abonnerings-funksjonene initialiseres. Når det er gjort publiseres en melding til ROS-console om at noden er klar til å ta imot koordinater.

Når noden mottar en liste med koordinater av meldingstypen

`InspectionLatLongArray`, aktiveres funksjonen `LatLongArray_cb`, en `callback`-funksjon som er tilknyttet et spesifikt `topic`. Deretter sendes første koordinat i listen til noden `/gps_conversion_node` for konvertering til robotens koordinatsystem. Konvertert koordinat mottas så i `callback`-funksjonen `pose_cb` som markerer starten på rutefølgingsprosessen. Først sjekkes det om siste mål er nådd basert på lengden av innkommende liste, en sjekk som ikke er relevant før koordinat nr. 2. Koordinaten behandles så av funksjonen `movebase_client` hvor koordinaten konverteres til datatypen `MoveBaseGoal` som benyttes av `move_base`.

Koordinaten sendes så til `MoveBase`, hvor det først blir kontrollert om destinasjonen ligger innenfor robotens globale `costmap`, og basert på utfall splittes programflyten videre. Hvis målet er innenfor globalt `costmap` betyr det at roboten har et kartgrunnlag som den kan basere ruteplanlegging på. Selv om kartet tillates å være tomt betyr det at oppdukkende hinder kan representeres innenfor kartets nåværende rammer.

- *Status = 5*: Hvis det ikke finnes i orden, blir rutefølgning av nåværende koordinat kansellert og `MoveBase` returnerer via en forhåndsdefinert statusmelding at gjeldende koordinat ble avvist. Programmet fortsetter så ved å forsøke å gå til neste mål i listen.
- *Status = 4*: Målet er innenfor `costmap`, og robot har startet rutefølgning men ble kansellert av ruteplanlegger fordi den ikke fant en mulig vei til målet blant hindrene i `costmap`. Meldingen vil også forekomme dersom ruteplanlegger detekterer at roboten *oscillerer* mellom punkter og ikke kommer seg noen vei. Det er ansett som en alvorlig feil og resulterer i fullstendig stans av rutefølgning. Ny liste med koordinater må sendes for å starte på nytt.
- *Status = 2*: Målet er innenfor `costmap`, robot har startet rutefølgning men ble kansellert av bruker. Roboten vil da gå til målet før den stanser og venter på nye koordinater.
- *Status = 8*: Samme som 2, men ble kansellert før rutefølgning rakk å starte. Venter på nye koordinater.
- *Status = 3*: Målet er innenfor `costmap` og roboten har nådd målet. Det sendes melding til `test_node` for logging av at nåværende mål er nådd, før målteller telles opp og det sjekkes om siste mål i ruten er nådd. Hvis `InspectionLatLongArray`

inneholdt informasjon om at video-opptak skal starte gjøres det og deretter ventes det en spesifisert tid før neste mål sendes til konvertering, og hele prosessen starter på nytt.

MoveBase sender meldinger parallelt om hva som foregår, samt gjenværende distanse for nåværende mål til ROS-console via callback-funksjonen `feedback_cb`.

Beskrivelse av `gps_conversion_node`

For å konvertere koordinater oppgitt i lengde- og bredde-grader til koordinater i robotens koordinatsystem `odom`, er det laget en node i C++. Den benytter en funksjon fra ROS-pakken `robot-localization` kalt `LL2UTM`[33] som konverterer lengde- og bredde-grader til UTM koordinater.

Noden initialiseres samtidig som `move_base_client` startes og venter deretter på et enkelt koordinat på topic `/gps_conversion_node/goal_input`. Når den mottar et koordinat, konverteres koordinaten til UTM-koordinatsystemet. Koordinaten er nå oppgitt i meter, slik at det kun gjenstår å forflytte koordinaten fra referansekoordinatsystemet `utm` til `odom`, som gjøres i funksjonen `UTM2Odom`. Til slutt publiseres koordinaten tilbake til `move_base_client`, slik at roboten kan starte rutefølgning med koordinater i sitt koordinatsystem.

3.1.2 ROS-meldinger

Det finnes ingen standard meldingstyper i ROS som kan bestå av flere koordinater av typen lengde- og bredde-grader. I tillegg er det i denne oppgaven behov for å sende ekstra informasjon sammen med koordinater, som ventetid per punkt, `yaw` og start eller stopp av video-opptak. Det er derfor opprettet egendefinerte meldingstyper som kan ta seg av slik funksjonalitet.

InspectionLatLongArray

Meldingstypen `InspectionLatLongArray` (*norsk: tabell med lengde- og bredde-grader for inspeksjon*) er en tabell som kan inneholde N elementer av typen `inspectionlatlongs`. Den er bygget opp på følgende måte:

- `InspectionLatLongArray[N]`
 - `inspectionlatlongs`
 - * `inspectionlatlong`
 - `float32` `idletime` (*ønsket ventetid for dette koordinatet*)
 - `bool` `record` (*boolsk variabel som starter opptak når høy*)
 - `sensor_msgs/NavSatFix` `navsatfix` (*standardmelding, len-og br.gr*)
 - `latitude` (*ønsket breddegrad*)
 - `longitude` (*ønsket lengdegrad*)
 - `yaw` (*ønsket yaw*)
 - * `header` (*standard ROS metainformasjon*)
 - `frame_id` (*referansekoordinatsystem*)
 - `time_stamp` (*tid*)

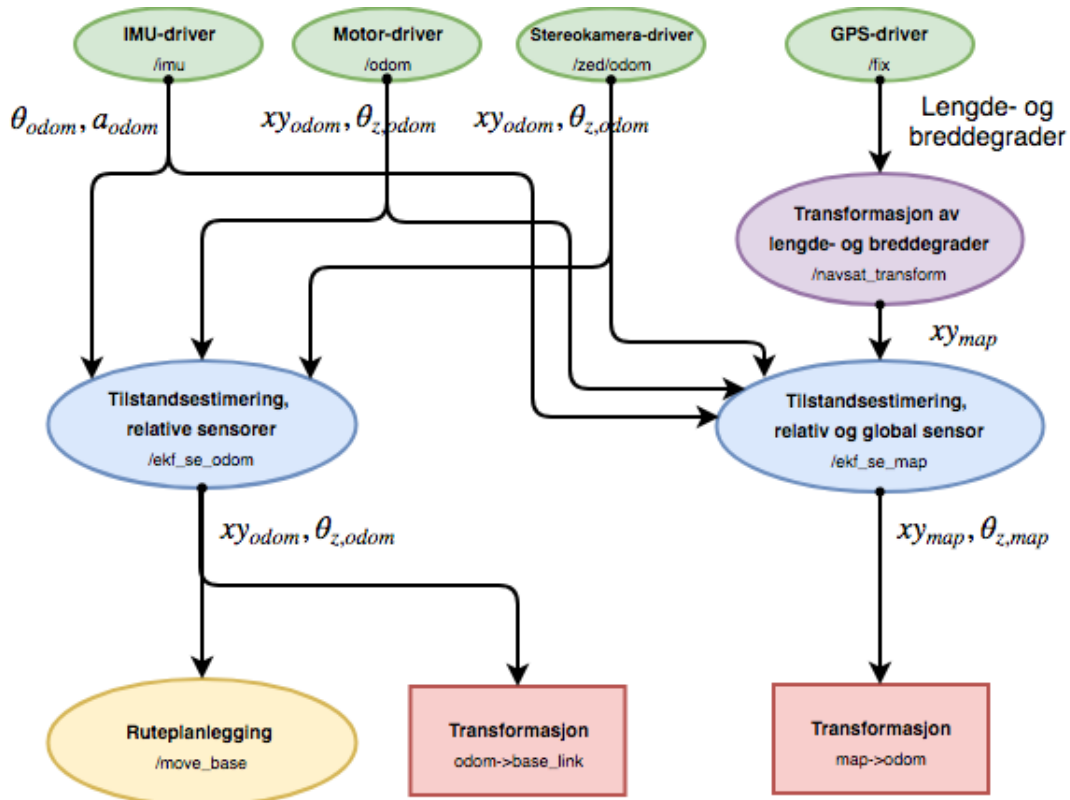
Det er også nødvendig med en ny meldingstype for koordinater definert i robotens koordinatsystem, `InspectionPoseArray` (*norsk: tabell med positur for inspeksjon*):

- `InspectionPoseArray[N]`
 - `inspectionposes`
 - * `inspectionpose`
 - `float32` `idletime` (*ønsket ventetid for dette koordinatet*)
 - `bool` `record` (*boolsk variabel som starter opptak når høy*)
 - `geometry_msgs/Pose` `pose` (*standardmelding for positur*)
 - `position [x,y,z]` (*ønsket posisjon i kartesiske koordinater*)
 - `orientation [x,y,z,w]` (*ønsket rotasjon i kvaternioner*)
 - * `header` (*standard ROS metainformasjon*)
 - `frame_id` (*referansekoordinatsystem*)
 - `time_stamp` (*tid*)

3.1.3 Sensorfusjon

For å estimere egen posisjon benytter roboten en kombinasjon av flere sensorer. Anvendt konfigurasjon for tilstandsestimering illustreres i figur 27

Oppsett



Figur 27: Oversikt over oppsett for sensorfusjon. Grønne figurer er sensordrivere, blå er tilstandsestimerings-noder, rød er transformasjoner mens gul er ruteplanleggingsnode.

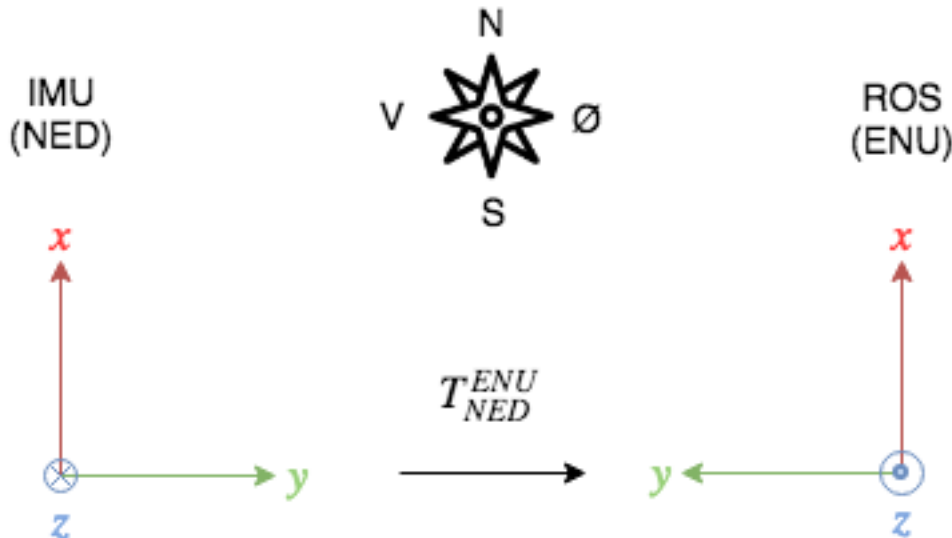
Tilstandsestimering foregår i to noder; `ekf_se_odom` og `ekf_se_map` (*Extended Kalman Filter State Estimation map/odom: utvidet Kalman-filter, tilstandsestimering for map/odom*). Som navnene antyder, er estimering delt opp for hvert referansekoordinatsystem. Noden som produserer den *kontinuerlige* transformasjonen `odom` \rightarrow `base.link` tar inn sensordata fra de *relative* sensorene IMU, stereokamera, og motordometri. Tilsvarende, for den ikke-kontinuerlige transformasjonen `map` \rightarrow `odom` tar den inn sensordata fra den *globale* og *diskrete* GNSS-mottakeren, men tar i tillegg inn data fra de *relative* sensorene. Det medfører at hver måling fra GNSS-mottaker (1 Hz) blir ansett som et nytt utgangspunkt for de *relative* sensorene, som derfor aldri får tid til å introdusere feil av betydning.

3.2 Modifisering av eksisterende noder

For å oppnå ønsket funksjonalitet har det vært nødvendig å gjøre endringer i kilde-koden for allerede eksisterende noder.

3.2.1 IMU driver

IMU-driver[36] brukt i denne oppgaven følger koordinatsystemkonvensjonen *NED* (*North East Down*), og er derfor ikke direkte kompatibel med ROS som benytter standarden *ENU* (*East North Up*). For å omgå problemet var det nødvendig å sette inn en statisk transformasjon i kilde-koden til ROS-driveren for IMU. Figur 28 illustrerer problemstillingen.



Figur 28: NED og ENU

Utfra figuren er det enkelt å se at transformasjonen T_{NED}^{ENU} tilsvarer en rotasjon om x-aksen med 180° .

ROS krever også at når IMU peker rett *øst* skal den rapportere 0° rotasjon om z-aksen. Siden IMU følger den tradisjonelle konvensjonen for kompasskurs med *nord* tilsvarende 0° , må også det kompenseres for. Det gjøres ved å sette parameteren `navsat_transform_node/yaw_offset` til $\frac{\pi}{2}$, som resulterer i en rotasjon om z-aksen med 90° . Det endelige koordinatsystemets retninger er vist i figur 29.



Figur 29: Resulterende koordinatsystem, IMU

3.2.2 Konvertering av dybdebilde til laserscan

Ved konvertering av dybdebilder til `laserscan` benyttes en fast bilderad fra bildet, slik at informasjonen som blir sendt til ruteplanlegger alltid "ser" scenen i en fast høyde. Verdien av *skann-høyden* har vist seg å være noe problematisk siden stereokameraet er montert nærme bakken, som betyr at en stor del av bildet består av underlaget. Det resulterer i at "falske" hindringer fra underlaget blir detektert og dukker opp i `costmap` som faktiske hindringer, noe som forekommer hyppig ved kjøring i selv svakt kupert terreng. Det er problematisk siden ruteplanlegger da forsøker å unngå hindringer som i virkeligheten ikke eksisterer.

For å forhindre slik oppførsel ble det gjort endringer i kildekoden for ROS-pakken `depthimage_to_laserscan`[37] slik at *skannhøyden* får en verdi som tilsvarer en bilderad høyere i bildet. I praksis tilsvarer det å heve kameraet høyere opp i forhold til bakkenivå.

3.3 Parametre med særlig betydning

Under konstruksjon og konfigurasjon av systemet er det identifisert enkelte parametre som har spesielt stor betydning for oppførsel, og er derfor viktig å gi fornuftige verdier. Beskrivelsen av parametrene er delt opp etter hvilke delsystemer de påvirker.

Listene i delkapitlet angir først navn på konfigurasjonsfiler, og deretter parameternavn.

3.3.1 Navigasjon

For navigasjon finnes det hundrevis av parametre som påvirker oppførselen i større eller mindre grad. Mange av parametrene er forhåndsinnstilt, men siden ROS er et generelt operativsystem som skal fungere med mange typer roboter er det nødvendig å gjøre tilpasninger for det spesifikke oppsettet benyttet i denne oppgaven.

Referansekoordinatsystem for `costmap`

Parametrene i dette avsnittet har til felles at de påvirker hvilket referansekoordinatsystem ulike deler innen navigasjon skal utføres.

Parameteren `global_frame` for `global_costmap` er av stor betydning for hvordan selvstendig navigasjon fungerer. Den bestemmer hvilket referansekoordinatsystem som benyttes av kartet, og her er det viktig å kjenne til hvilke andre funksjoner som benytter de ulike koordinatsystemene.

- `local_costmap_params`
 - `global_frame`: `odom` (alltid)
- `global_costmap_params`
 - `global_frame`: `map` eller `odom`, se beskrivelse.

`teb_local_planner_params`

- `map_frame`: samme som for `global_cost_map`

For `local_costmap` benyttes alltid `odom` som referanse. `Global_costmap` kan benytte begge, men det er viktig å vite hvordan systemet oppfører seg for hver av dem. Når et målpunkt publiseres i ROS vil det alltid referere til `global_costmap` sin `global_frame`. Det vil si at dersom `map` velges, vil målpunktet være forankret i referansekoordinatsystemet `map`. Det har den fordelen at det alltid vil være *verdensorientert*, som betyr at det følger transformasjonen som kommer fra den globale sensoren `map` → `odom`, som generelt stemmer best med virkeligheten over tid. Det

kan imidlertid oppstå en ulempe hvis den globale sensoren er unøyaktig, og dermed har en tendens til å produsere diskrete hopp i sitt estimat. Det kan føre til at roboten aldri vil nå målet fordi det befinner seg i en ny posisjon (flere meters forskjell) for hver oppdatering (typisk 1 Hz). Er sensoren derimot nøyaktig, og dermed produserer et estimat med korte og sjeldne diskrete hopp, vil roboten nå målet samtidig som det er verdensorientert. Dette er den foretrukne måten å konfigurere systemet på, og er benyttet i denne oppgaven da den globale sensoren gir lav nok variasjon.

Hvis `odom` velges vil det publiserte målpunktet være forankret i referansekoordinatsystemet `odom`, som er basert på relative sensorer. Det har den fordelen at det aldri vil inneholde diskrete hopp, og dermed vil roboten alltid klare å nå målet. Ulempen er her at `odom` vil akkumulere feil fra relative sensorer og vil dermed ikke være verdensorientert, som fører til at roboten vil treffe målpunktet med en feil tilsvarende total opparbeidet feil i `odom`. Ved bruk av en global sensor med stor spredning kan dette likevel være en foretrukken løsning, og det kan teoretisk sett lages ekstra transformasjoner som benytter data fra `map`→`odom` til å gjøre målpunkt verdensorientert.

For begge valgene er det i tillegg kritisk å påse at parameteren `map_frame` for lokal planlegger (vist nederst i listen over) er satt til tilsvarende referansekoordinatsystem som `global_costmap`. Den bestemmer hvilket referansekoordinatsystem den beregnede rutens koordinater publiseres i.

Costmap og maskinsyn

Parametrene i dette avsnittet påvirker terskelen for hvilke observasjoner som legges inn i og klareres fra `costmap`, samt hvor nært en observasjon påføres en kostnad. Innstillingene er felles for både lokalt og globalt `costmap`.

- `common_costmap_params` (*felles parametre for kostkart*)
 - `obstacle_layer` (*hindringslag*)
 - * `obstacle_range` (*hindringsavstand*): 3-8 m
 - * `raytrace_range` (*strålesporingsavstand, alltid størst*): 4-9 m
 - `inflation_layer` (*utvidelse av grenser for hindringer*)
 - * `cost_scaling_factor`: 15
 - * `inflation_radius`: 0.5 m

Innstillingene over må ha fornuftige verdier for å oppnå akseptabel oppførsel for roboten i situasjoner hvor den må unngå hindre. Det er gjennom prøving og feiling funnet at stereokamera ikke bør legge inn hindringer for større avstander enn 3 m i omgivelser som inneholder mange hindringer. I mer åpne landskap med større og færre hindringer kan `obstacle_range` økes til rundt 8 m.

De resterende parametrene påvirker objektets kost, og er her satt relativt lavt slik at roboten har god fremkommelighet i trange omgivelser.

- `camera.launch` (*innstillinger for stereokamera*)
 - `confidence` (*tiltro til observasjon*): 70 %

I likhet med modifikasjonen gjort i 3.2.2 er parameteren over viktig å sette riktig for å unngå at falske hindringer dukker opp i `costmap`. Den bør settes så lavt som mulig samtidig som det passes på at små hindringer ikke forsvinner.

3.3.2 Ytelse og beregningsmessig belastning

Siden roboten er mobil har den begrenset tilgang til ressurser som prosesseringskraft og batterikapasitet. For å tilpasse prosessorbruk finnes det mange parametre som kan justeres uten negativ påvirkning på systemets oppførsel.

Ruteplanlegger

- `teb_local_planner_params`
 - `enable_homotopy_class_planning`: `false`

Den første parameteren finnes i ruteplanleggerens innstillinger, og har stor påvirkning på reell ytelse. `Homotopy class planning` er kort forklart en metode for ruteplanlegging som planlegger og evaluerer flere alternative ruter parallelt. Ved å deaktivere denne metoden er det ikke observert problemer ved planlegging av ruter, selv i trange og kompliserte rom. Når funksjonen er aktiv oppleves det derimot at systemet har svært lite ressurser til overs, som viser seg i form av at roboten stopper opp etter hver planleggingssyklus (hver 4. meter) i noen sekunder, før den kjører videre og stopper når neste syklus skal starte.

Oppdateringshastighet

Parametrene i dette avsnittet går ut på å kontrollere hastigheten for sykliske beregninger.

- `global_costmap_params`
 - `update_frequency`: 1 Hz
 - `publish_frequency`: 0.5 Hz
- `local_costmap_params`
 - `update_frequency`: 5 Hz
 - `publish_frequency`: 2 Hz
 - `width (bredde)`: 5.5 m
 - `height(høyde)`: 5.5 m
 - `resolution(oppløsning)`: 0.1 m
- `ekf_params`
 - `ekf_se_map: frequency`: 30 Hz
 - `ekf_se_odom: frequency`: 30 Hz

Andre parametre som har en åpenbar påvirkning på benyttet beregningskraft er oppdateringshastighet for beregninger som gjøres syklisk. Oppdateringshastighet for `costmap` er justert slik at de reflekterer sitt respektive bruksområdes natur, slik at informasjon om endringer i nærområdet for roboten oppdateres raskere enn for endringer i periferien. Tilsvarende er estimering av egen posisjon av høy betydning og nodene for tilstandsestimering har derfor en hurtig oppdateringsfrekvens. Posisjonssensorers oppdateringsfrekvens er enten satt tilsvarende eller maksimalt for respektiv sensor.

Omfanget av sykliske beregninger kan også begrenses via størrelsesparametrene for `costmap`. Nødvendig størrelse avhenger av bruksområde, robotens hastighet samt topografi, og er for denne oppgaven funnet ved utprøving.

Stereokamera

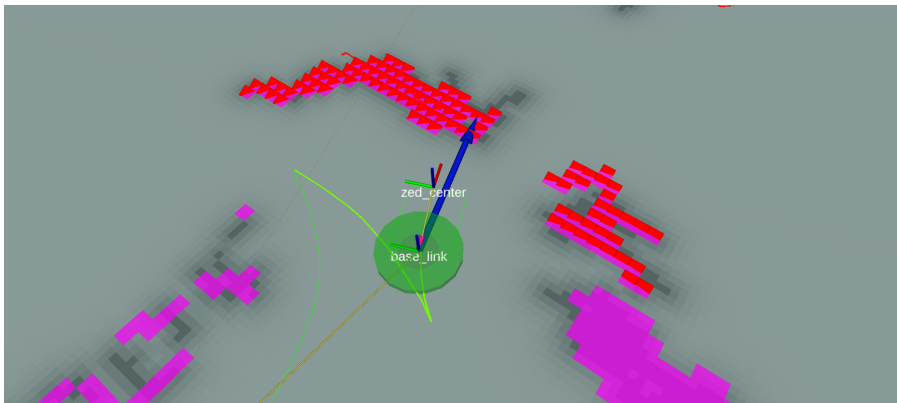
- camera.launch (Innstillinger for stereokamera)
 - resolution(*video-oppløsning*): 2 (2560x720 piksler)
 - quality (*kvalitet*): 1 (**performance**: prioritér scenens reelle størrelser foran estetisk filtrering)
 - framerate (*bildefrekvens*): 30 bilder/s

Kontinuerlig kalkulering av dybdebilder krever mye prosessorkraft, men prosessoren blir også avlastet ved at selve dybdeberegningen foregår på GPU (*grafikk-prosesseringsenhet*). Ved å begrense innstillingene som vist over oppleves det ingen negativ påvirkning på ytelse. Innstillingene ligger nok likevel i grenseland av hva systemet klarer å håndtere da ønsket bildefrekvens sjelden opprettholdes, og i stedet befinner seg rundt 20 bilder/s, som likevel er funnet å være tilfredsstillende.

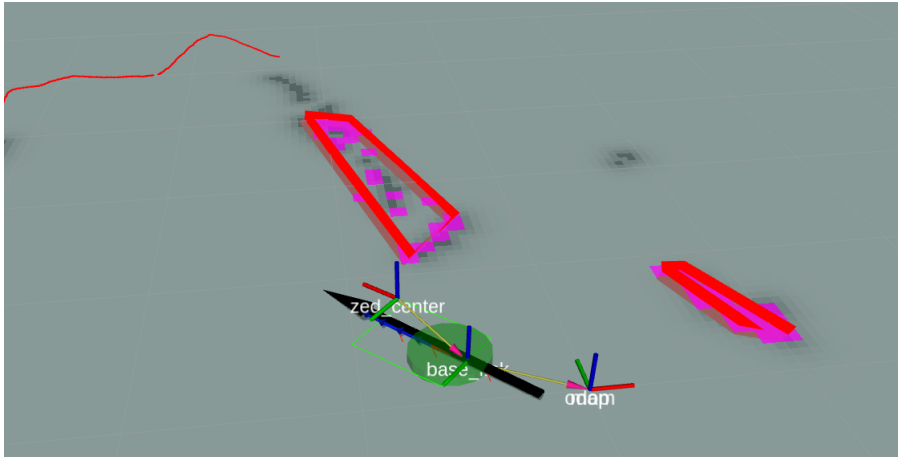
Konvertering av celler i costmap

- ROS-pakke: `costmap_converter`[38]

`Costmap_converter` er et ikke-standard programtillegg for `move_base` som konverter okkuperte celler fra `costmap` til approksimerte, enkle former. I utgangspunktet representeres objekter som celler med størrelse definert av `costmap` sin oppløsning (her 5 x 5 cm). Siden ruteplanlegger beregner avstand til hver enkelt celle i nærheten av en planlagt rute, kan det potensielt bli svært mange beregninger ved liten cellestørrelse. Programtillegget approksimerer derfor cellen som polygoner med et begrenset (justerbart) antall sider. Det gjør at antall distanser som må beregnes går *drastisk* ned i omgivelser med mange objekter, og så lenge polygonet ikke har for få sider slik at det ikke dekker de originale cellene, er det heller ingen negative konsekvenser.



Figur 30: Ruteplanleggers representasjon av relevante hindringer (rødt) som celler med størrelse 5x5 cm, før konvertering til polygoner. Lilla hindringer er lokalt `costmap` sin representasjon, mens grå er for globalt `costmap`. Grønn sirkel representerer ruteplanleggers modell av robot (merk at benyttet sirkel i etterkant er flyttet forover for bedre passform). Grønt rektangel er lokalt `costmap` sin modell av robot.



Figur 31: Ruteplanleggers representasjon av hindringer konvertert til polygon representerer den samme informasjonen på en mer effektiv måte.

- `teb_local_planner_params` (lokal ruteplanlegger)
 - `footprint_model` (størrelsesmodell): `circular` (sirkulær)

Parameteren over kan sees på som tilsvarende som forrige, men påvirker antall distanseberegninger som må gjøres for robotens form. Ved å representere robotens fysiske størrelse som en sirkulær modell som i figur 31 istedenfor rektangulær går det tapt noe informasjon ift. reell form, men det er ikke nødvendigvis kritisk for planlegging av rute så lenge modellens bredde er tilsvarende reell bredde. Til gjengjeld er det enkelt og effektivt å beregne distanser fra en sirkel.

4 Eksperimenter og resultater

Oppgavens mål er å konstruere et system som gjør at roboten kan operere semi-autonomt i et større område over tid. En grunnleggende og viktig forutsetning for slik funksjonalitet er at roboten kjenner egen posisjon presist. Det er med denne hensikt utformet eksperimenter for å verifisere at systemet klarer å opprettholde et nøyaktig posisjonsestimert over tid, hvor de individuelle sensorenes samt kombinerte ytelse vurderes. De initiale testene fokuserer på justere parametre og teste ytelse, mens videre testing dreier mer mot høynivå oppførsel som rutefølgning og unngåelse av hindre.

4.1 Evaluering av odometri: translasjon

Den eneste opprinnelige posisjoneringskilden for Spurv er motorodometri, og har derfor tidligere blitt kalibrert. Men, siden motorodometridata ikke har vært implementert i forbindelse med automatisk navigasjon, er det nødvendig med en grundig analyse. Andre odometrikilder som visuell odometri fra stereokamera samt sensorfusjon med motorodometri og IMU blir også vurdert. Stereokamera er på forhånd kalibrert så godt det lar seg gjøre, ved å tilpasse verdier som lysstyrke, kontrast, eksponeringstid og forsterkning samtidig som bildestrømmen vises.

For motorodometri forventes det god presisjon for translasjon langs x- og y-akse, mens forventningen til kameraets odometri er noe lavere grunnet lite homogene omgivelser. Sensorfusjon med IMU antas ikke å gi vesentlige forandringer for total translasjon, men derimot bedre evne til å skille mellom translasjon i x- og y-retning.

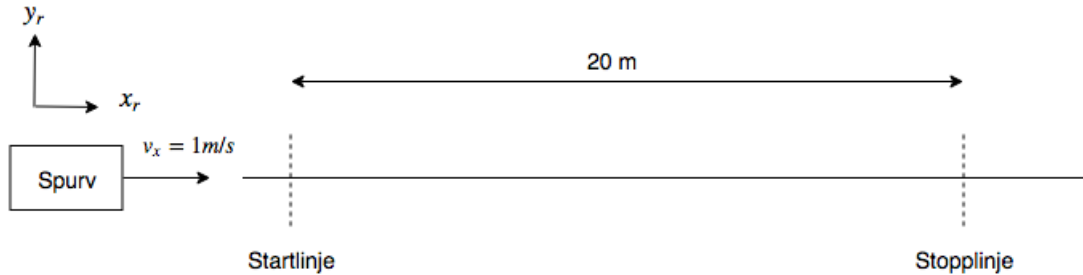
Testens hensikt er å vurdere sensorenes presisjon uten å blande inn kompliserende og mulig forstyrrende elementer som kan forekomme ved selvstendig kjøring. For å oppnå det styres fremdriftsmotoren ved å sette ønsket hastighet i m/s som så konverteres direkte til tilsvarende rotasjonshastighet i **erpm**. Siden parametrene som påvirker odometriens presisjon i realiteten bestemmer robotens hastighet, kan testen også sies å vurdere robotens evne til å oppnå ønsket hastighet.

De første eksperimentene har som mål å danne et bilde av hvor god nøyaktighet som kan forventes, og å se på muligheten for å forbedre ytelsen for motorodometri. Testen tar videre for seg de resterende resterende relative posisjoneringssensorene; visuell odometri fra stereokamera og sensorfusjon av motorodometri med IMU.

4.1.1 Testoppsett

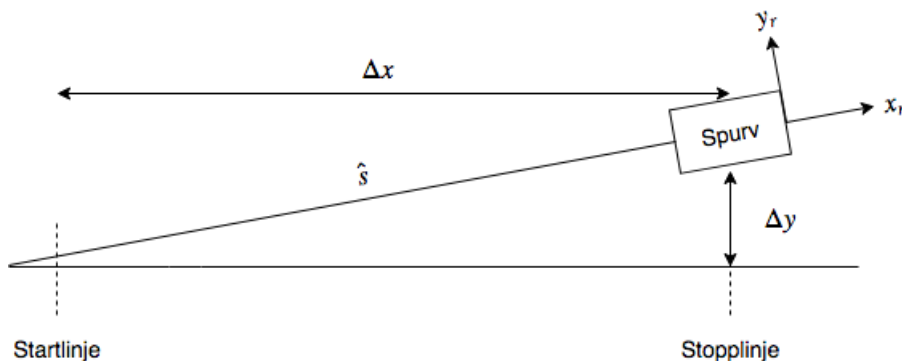
Testen gjennomføres i et utendørs testområde hvor en strekning på 20 m er målt opp. Robotens hastighet settes til 1 m/s og startposisjonen er ca 3 m i forkant av startlinjen, slik at roboten får mulighet til å akselerere til ønsket hastighet før måling starter. Dette sørger for at systemet er tilnærmet stabilt under testens loggfase. For

testen er det laget en enkel ROS-node som har en variabel som kan settes fra en ekstern enhet (laptop). Når den passerer startlinjen settes den høy og når den treffer stopplinja settes den lav. Variabelen kan deretter plottes sammen med odometrien slik at relevant data kan hentes ut enkelt og presist.



Figur 32: Prinsippkisse testoppsett, translasjon.

Siden roboten har noe unøyaktighet i drivverk forventes det at den ikke kjører beint fram, men trekker noe til siden. Denne sideforskyvningen måles, og faktisk kjørt avstand estimeres som hypotenusen i en trekant, $\hat{s} = \sqrt{\Delta x^2 + \Delta y^2}$, som vist i figur 33. s kalles her et estimat fordi reell bane er noe kurvet.



Figur 33: Resulterende testoppsett med unøyaktighet i drivverk.

For hver test logges det data fra 3 kilder etter følgende oppsett:

- Motorodometri
- Sensorfusjon av motorodometri og IMU
- Visuell odometri fra stereokamera

Figur 32 viser en prinsippkisse av testoppsett og listen under viser hvilken data som innhentes.

- Sensorkombinasjonens euklidske estimat av tilbakelagt avstand [m].

- Motorodometri: xy_{odom}
- Sensorfusjon, motorodometri og IMU: xy_{sf}
- Visuell odometri, stereokamera: xy_{vo}
- Δy : målt sideforskyvning ved stopplinje[m].
- Δt : faktisk tidsforskjell mellom startposisjon og stopposisjon [s].
- \hat{s} : estimert tilbakelagt euklidsk avstand [m] (beregnet).

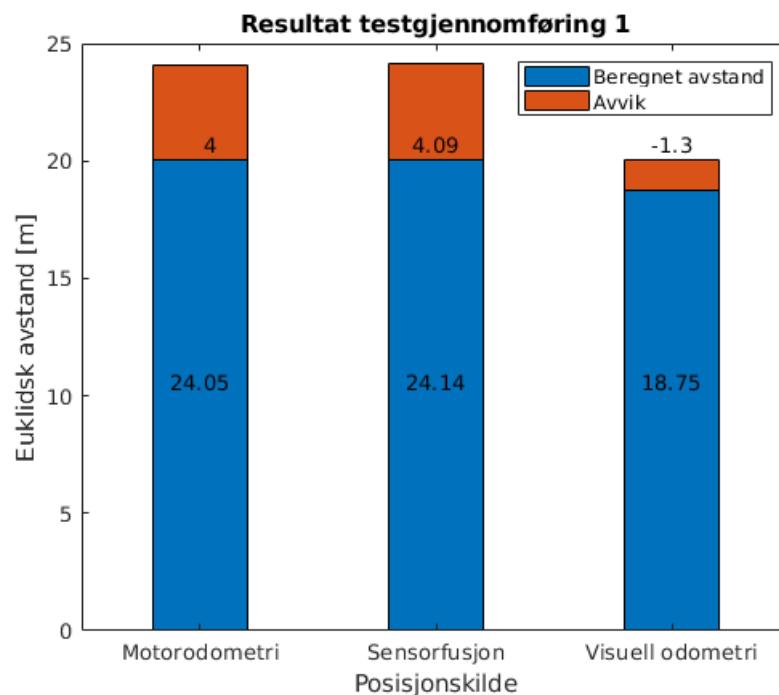
Testene er delt inn etter hvilken sensor som danner grunnlaget for posisjonering.

4.1.2 Resultat, translasjon

Testgjennomføring 1

Følgende resultat ble logget under første gjennomføring:

- x_{odom} : 24.05
- ω_{motor} : 4084
- $\Delta y = 1.4m$
- Δt : 23.97s



Figur 34: Resultat, test 1, faktisk kjørt avstand er 20.05 m

som gir

$$\hat{s} = \sqrt{\Delta x^2 + \Delta y^2} = \sqrt{20^2 + 1.4^2} = \underline{20.05m} \quad (24)$$

og feilen e :

$$e = x_{odom} - s = 24.05 - 20.05 = 4m$$

som gir en prosentvis feil på

$$e_p = \frac{4}{20} = 0.20 = 20\%$$

Robotens estimat av tilbakelagt avstand tilsier at den har kjørt 24 m når den egentlig bare har kjørt 20 m. Det er 4 m for langt, noe som skyldes en for lav verdi i konverteringskonstanten `speed_to_erpm_gain`.

Justering av parameter

Data fra testen kan brukes til å regne ut en mer passende verdi for konverteringskonstanten. Fra kapittel 2.2.1 hentes følgende sammenheng:

$$\dot{x}_r [\text{m/s}] = \frac{\omega_{motor} [\text{rpm}]}{K_{SpeedToRpm} [\text{rpm/m/s}]} \quad (25)$$

Fra testdata kan reell gjennomsnittshastighet \dot{x}_r estimeres ved:

$$\hat{\dot{x}}_r = \frac{s}{\Delta t} = \frac{20m}{23.97s} = 0.83 \text{ m/s} \quad (26)$$

For å finne ny verdi snus ligning (25) og testresultat-data settes inn:

$$K_{SpeedToRpm} = \frac{\omega_{motor} [\text{rpm}]}{\dot{x}_r [\text{m/s}]} = \frac{4084}{0.83} = \underline{4920 \text{ rpm/m/s}} \quad (27)$$

Den nye verdien for $K_{SpeedToRpm}$ er som forventet høyere enn den gamle, som betyr at ved ønsket hastighet på 1 m/s vil motoren nå rotere med 4920 erpm istedenfor den gamle verdien på 4084 erpm. En ny test gjennomføres så for å validere den nye verdien.

Testgjennomføring 2: nye parameter

Følgende resultat ble logget under andre gjennomføring:

- $x_{odom} : 20.21m$
- $\omega_{motor} : 4920erpm$
- $\Delta y = 1.8m$
- $\Delta t : 20.09s$

som gir en estimert lengde på:

$$\hat{s} = \sqrt{\Delta x^2 + \Delta y^2} = \sqrt{20^2 + 1.8^2} = \underline{20.08m}$$

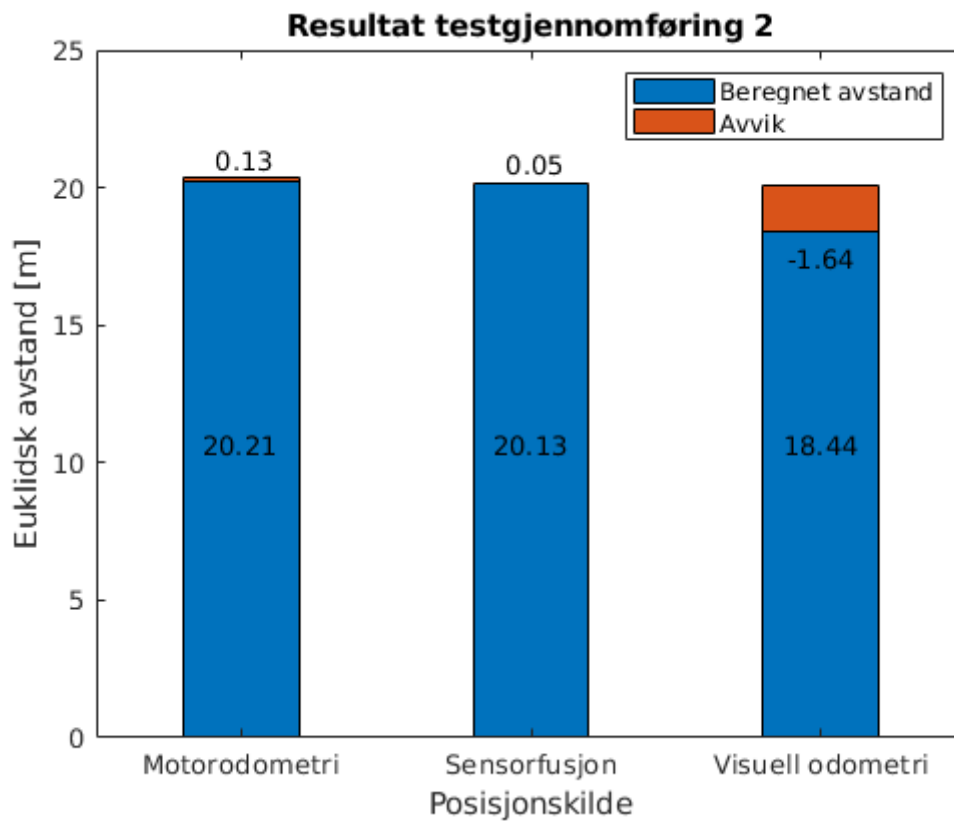
og feilen e:

$$e = x_{odom} - s = 20.21 - 20.08 = 0.13m$$

som gir en prosentvis feil på:

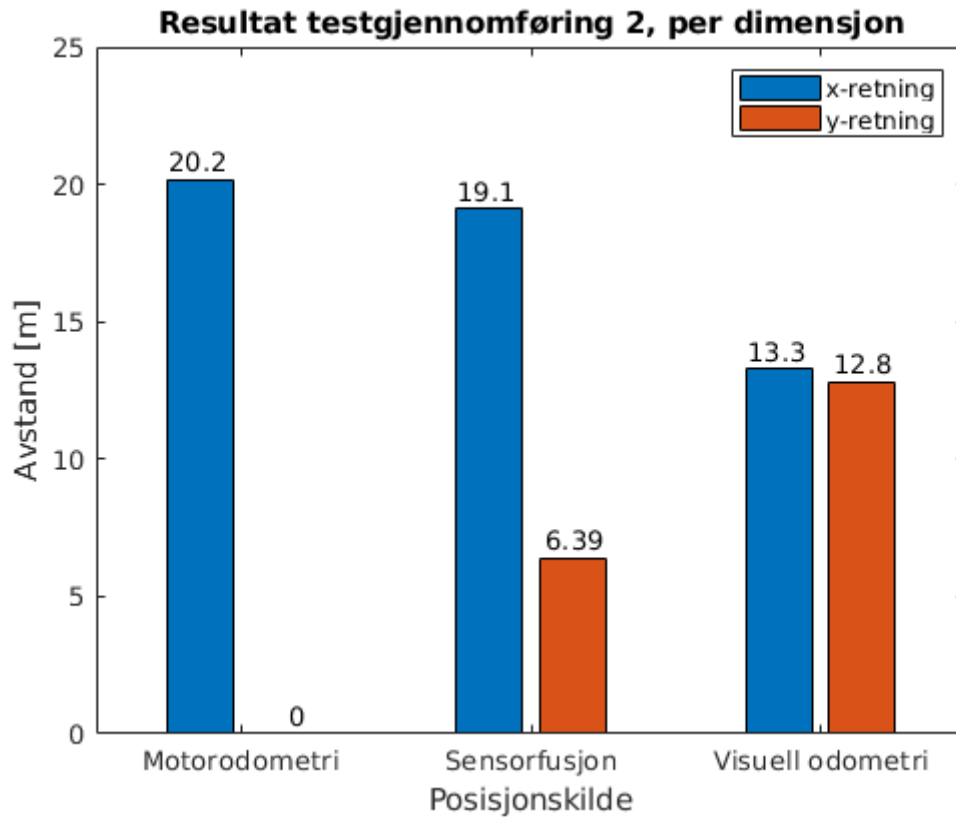
$$e_p = \frac{0.13}{20.08} = 0.0065 \approx 1\%$$

For alle sensorer:



Figur 35: Resultat test 2, euklidsk avstand

Ved å bare se på figuren over ser det ut som om resultatet for motorodometri og sensorfusjon er tilnærmet identisk, men forskjellen finnes ved å se på tilbakelagt avstand per dimensjon, se figur 36.



Figur 36: Resultat test 2 som i forrige figur, men viser nå distansen for hver dimensjon.

Figur 36 viser at data fra motorodometri indikerer at robot kjører et rett strekk langs x-akse, mens resultat for sensorfusjon viser at distanse fordeles på de to dimensjonene. Grafen for visuell odometri viser at distansen er høy langs begge akser.

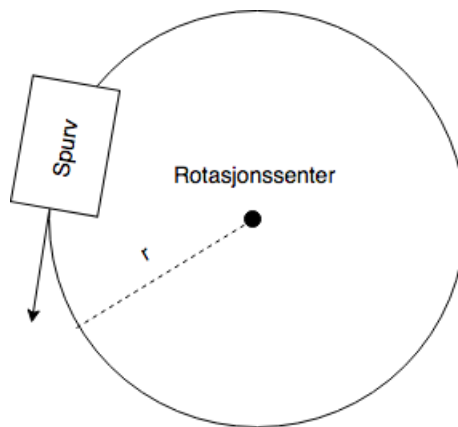
4.2 Evaluering av odometri: sving

Parameteren som i denne testen skal evalueres påvirker robotens evne til å bestemme utslagsvinkel for servomotorer gitt en vilkårlig ønsket svingvinkel for roboten. Siden den samme parameteren brukes til estimering av robotens posisjon, vil det også påvirke nøyaktighet for posisjonsestimater. For selvstendig kjøring er det nødvendig å kjenne minste mulige svingradius, i tillegg til forholdet mellom hjulenes svingvinkel og robotens svingradius slik at baneplanlegger kan ta nødvendige hensyn.

4.2.1 Testoppsett

Testen gjennomføres ved å kjøre roboten med full sving slik at tilbakelagt bane danner en sirkel som representerer minste mulige svingradius. Sirkelens radius måles deretter med målebånd for sammenligning med robotens estimat. For denne testen inngår ikke robotens hastighet som en relevant parameter, og settes derfor til en vilkårlig, lav verdi. Figur 37 illustrerer testens oppsett.

Den første testen evaluerer den opprinnelige verdien for parameteren $\phi_{ServoCommand}$, før det kalkuleres ny verdi som vurderes i påfølgende test. Videre gjennomføres det en tilsvarende test med full sving motsatt vei, før det gjøres en gjennomføring hver vei med mindre svingvinkel for å verifisere at korrigert parameter stemmer for andre svingradier.



Figur 37: Testoppsett, evaluering av svingradius hvor estimert og målt radius sammenlignes.

Følgende parameter skal vurderes:

- $K_{SteeringToServo}$: parameteren som bestemmer svingutslag [utslag / rad].

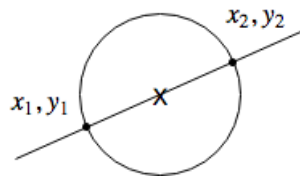
Følgende variabler logges:

- Distanser
 - x_{odom}, y_{odom} : motorodometriens estimat av tilbakelagt avstand i x og y-planet [m].
 - x_{vo}, y_{vo} : stereokameraets estimat av tilbakelagt avstand i x og y-planet [m].
 - x_{sf}, y_{sf} : sensorfusjonens (motorodometri og IMU) estimat av tilbakelagt avstand i x og y-planet [m].
- Annet
 - $\phi_{ServoCommand}$: Utslag servomotor (for verifikasjon av satt parameter).

Følgende fysiske målinger logges:

- $r_{m\ddot{a}lt}$: sirkelens radius funnet med målebånd [m].

Robotens estimerte svingradius finnes så ved å regne den euklidske avstanden mellom to punkter i xy_{odom} som er skjæringspunktet mellom et linjestykke som går igjennom senter og sirkelens omkrets, illustrert i figur 38.



Figur 38: Beregning av estimert svingradius

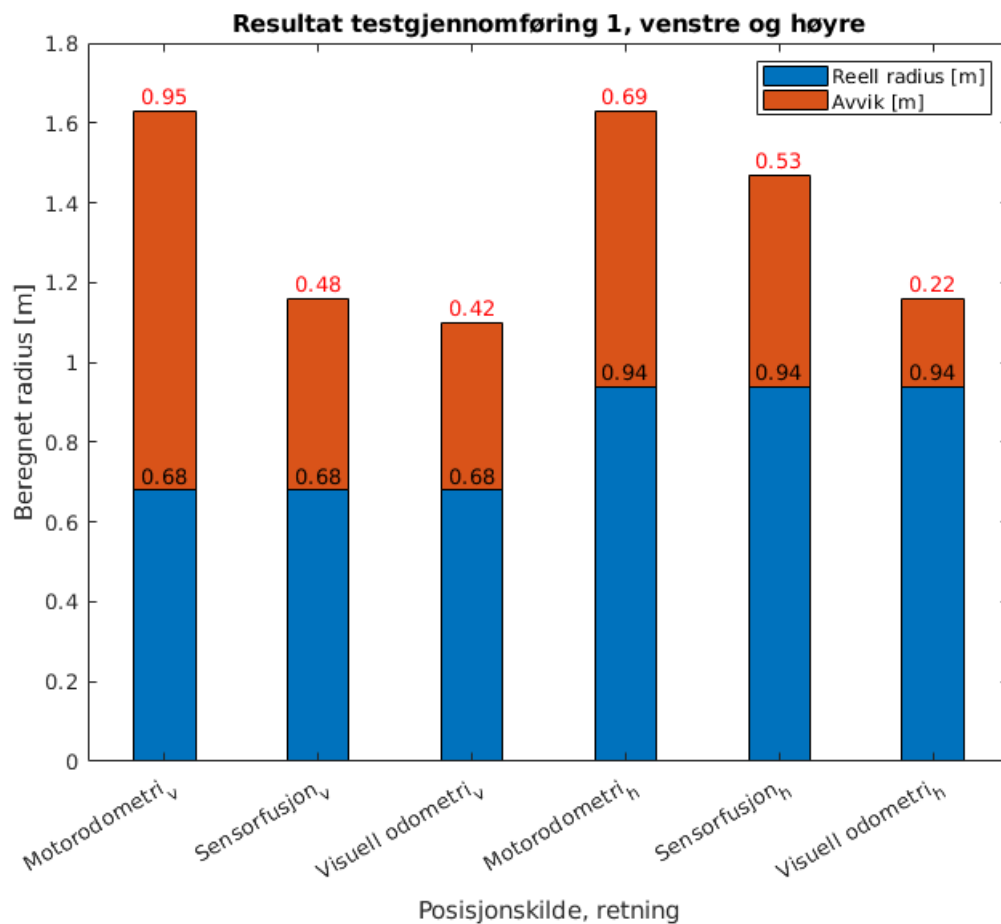
4.2.2 Resultat

Testgjennomføring 1, venstre- og høyre-sving

Testparametre:

- $\phi_{ServoCommand}$:
 - 0.75: tilsvarer maksimal venstresving
 - 0.25: tilsvarer maksimal høyresving
- $K_{SteeringToServo}$
 - 1.1204: opprinnelig verdi

Figur 39 viser resultat logget under første gjennomføring:



Figur 39: Resultat, første gjennomføring. Avvik mellom reell og estimert radius i rødt, reell radius i svart.

Her er det tydelig at $K_{SteeringToServo}$ har en for høy verdi, som medfører at roboten estimerer radius til å være større enn den i realiteten er.

Justering av parameter

Data fra testen kombinert med ligninger fra foroverkinematikk og konvertering av størrelser kan brukes til å finne verdier som gjør at estimatet stemmer bedre med virkeligheten. Siden radius varierer utfra hvilken side som svinges til, brukes den største av de to, $r = 0.94m$, samt at det legges til litt rom for variasjon, $r = 1m$. Fra kapittel 2.1.4 hentes ligning (5) og (11):

$$\tan(\phi) = \frac{L}{r} \quad (5)$$

$$\phi = \frac{\phi_{ServoCommand} - Offset_{SteeringToServo}}{K_{SteeringToServo}} \quad (11)$$

Ved å sette ligning (11) inn i ligning (5) kommer følgende sammenheng fram:

$$\tan\left(\frac{\phi_{ServoCommand} - Offset_{SteeringToServo}}{K_{SteeringToServo}}\right) = \frac{L}{r}$$

og ved å løse den mhp. konverteringskonstanten finnes

$$K_{SteeringToServo} = \frac{\phi_{ServoCommand} - Offset_{SteeringToServo}}{\tan^{-1}\left(\frac{L}{r}\right)} \quad (28)$$

For å regne ut en ny konstant settes det inn data fra testen, faktisk radius: $r = 1m$

$$K_{SteeringToServo} = \frac{0.75 - 0.5}{\tan^{-1}\left(\frac{0.37m}{1m}\right)} = 0.706 \text{ [utslag / rad]}$$

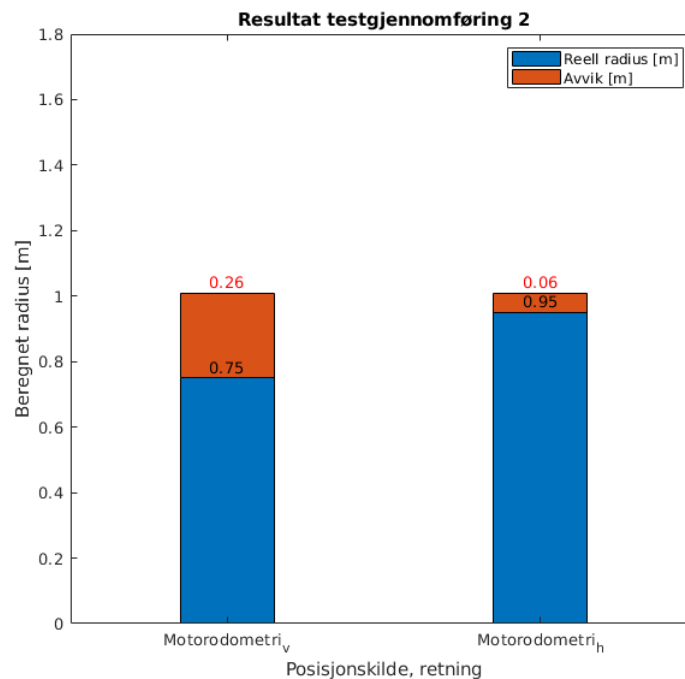
Testgjennomføring 2, justert parameter, venstre- og høyre-sving

Den neste testen har til hensikt å evaluere eventuell forbedring med justert parameter. Parameteren påvirker bare motorodometri, de resterende sensorene er derfor utelatt i denne testen.

Testparameter:

- $K_{SteeringToServo}$
 - 0.704: justert verdi

Figur 40 viser resultat fra andre gjennomføring:



Figur 40: Resultat, justert parameter

Resultatet viser at avvik mellom reell og estimert svingradius er redusert fra 0.95m til 0.26m, og fra 0.69m til 0.16m for maksimal sving hhv. venstre og høyre. Det er likevel et relativt stort avvik mellom svingradiene på 20cm.

For å kunne si mer om hvor mye dette påvirker resultatet, gjøres det en test med mindre svingutslag.

Test 3: mindre svingutslag

For å sjekke om parameteren fungerer godt over en større del av servomotor-utslaget

område, gjennomføres det en test med parametrene beskrevet under, som gir et mindre svingutslag og dermed større svingradius.

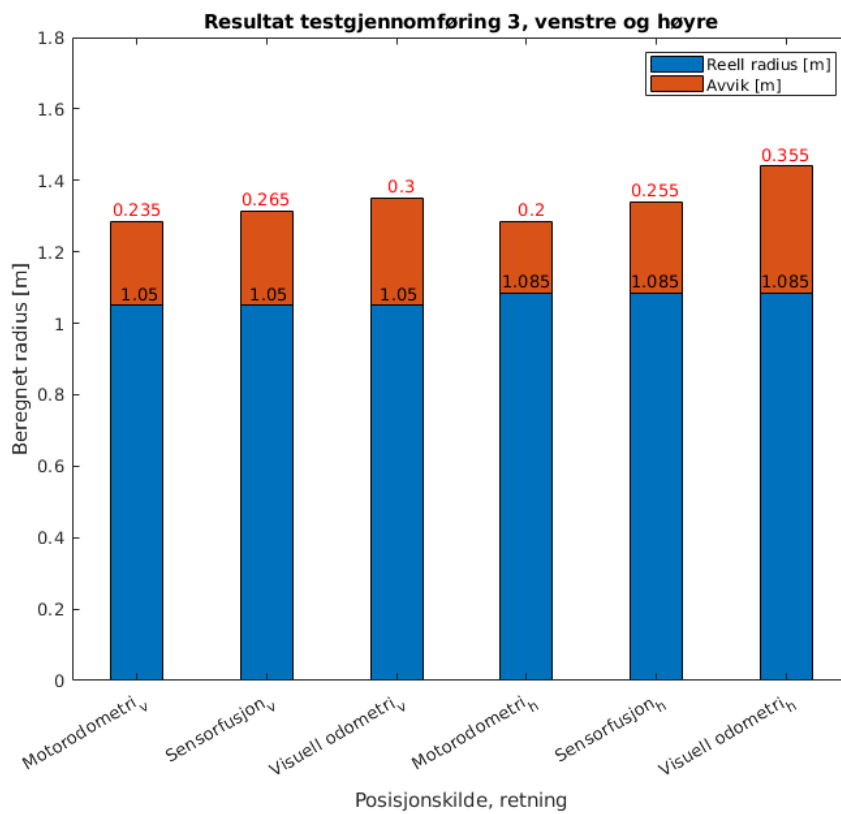
- $\phi_{ServoCommand}$:

- 0.70

- 0.30

- $K_{SteeringToServo}$

- 0.704



Figur 41: Resultat, test 3

Resultatet viser at ved mindre svingutslag er den negative effekten som fører til ulik svingradius mindre fremtredende. Det registreres likevel til et større generelt *avvik* mellom reell og estimert radius.

4.3 Navigasjon ved relativ posisjonering

Testene i denne seksjonen skal kartlegge hvor godt roboten navigerer selvstendig ved bruk av relative sensorer.

4.3.1 Testoppsett

Alle testene i denne seksjonen er gjennomført utendørs i et område med en størrelse på ca 60 x 17 m, vist i figur 42. Testene er i hovedsak delt i to, mens videre nedbrytning av testene er beskrevet under. Parametrene som ble funnet i testkapitlet 'Evaluering av odometri' er brukt videre her.

Alle tester benytter sensorfusjon med IMU og motorodometri som tilbakekobling til ruteplanlegger. Lokal ruteplanlegger er konfigurert med en måltoleranse på 0.3 m og 0.5 rad (ca 30°).



Figur 42: Tilgjengelig utendørs testområde, benyttet for alle navigasjonstester i denne oppgaven.

Punktene er hentet fra `google maps-kart`[39], og er derfor opprinnelig definert i lengde- og bredde-grader. De blir på forhånd transformert til kartesiske koordinater, relativt til første rutepunkt, slik at det enkelt kan brukes den samme ruten for både relative og globale tester. Det gjøres på følgende måte:

Først transformeres lengde- og bredde-grader til utm, ved bruk av matlab-funksjonen `ll2utm`[34]

$$P_{utm} = ll2utm(P_U)$$

Deretter beregnes relative koordinater ved å trekke fra koordinatene til det første punktet i ruten

$$P_{rel} = P_{utm} - P_{utm}(1, 1)$$

Alle punkter blir da relativ i forhold til første punkt i ruten, med himmelretninger intakt.

Ved å sørge for at roboten initieres når den står plassert i punkt 1 vil koordinat-systemenes translasjon stemme overens, mens himmelretninger blir tatt hånd om av magnetometeret i IMU.

Test 1: fra punkt til punkt



Figur 43: Testoppsett 1

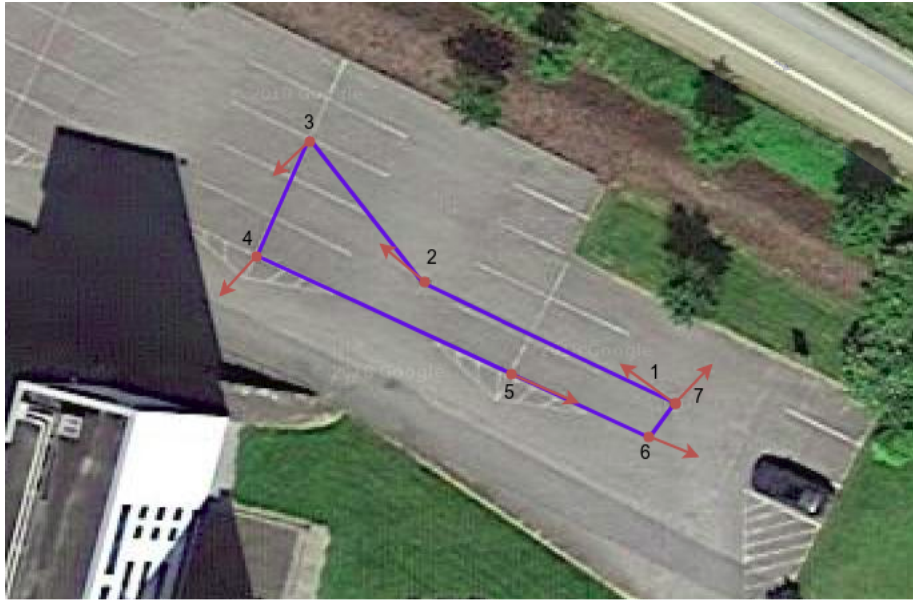
Hensikten med denne testen er å finne ut med hvor god presisjon roboten klarer å forflytte seg til et punkt ved selvstendig navigering. Den har likheter med tester gjennomført i "Evaluering av odometri", men det er ikke nødvendigvis selvsagt at resultatene gir tilsvarende presisjon ved selvstendig navigasjon. Testen gjennomføres ved å sende roboten fra startpunkt til stoppunkt som befinner seg ca. 10 m unna. Punktet sendes til roboten i lokalt koordinatsystem *odom*. Testbane illustreres i figur 43.

Listen under viser hvilke sensorer som rapporterer posisjon til lokal ruteplanlegger per test.

- Sensoroppsett nr. 1
 - Motorodometri (translasjon og vinkel)
- Sensoroppsett nr. 2
 - Sensorfusjon, motorodometri (translasjon) og IMU (vinkel).

Det gjøres fire gjennomføringer per sensor, hvor robotens distanse fra stoppunkt måles for hver gang.

Test 2: Rute



Figur 44: Rute, relativ test 2

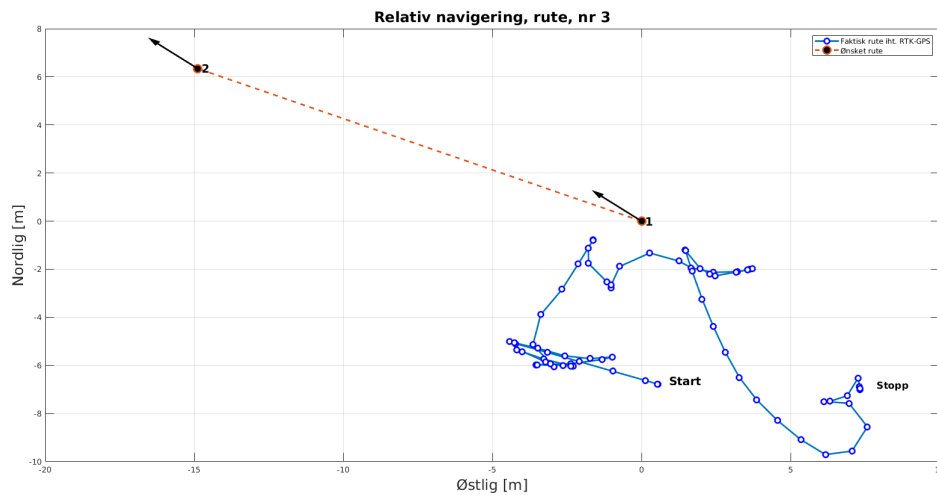
Den andre testen går ut på å sende roboten langs en rute som består av de 7 punktene vist i figur 44. Hensikten er her å fremprovosere eventuell unøyaktighet ved kjøring over lengre distanser med svinger. Ved å sende roboten i en relativt avansert bane som har samme endepunkt som startpunkt, er tanken at det faktiske endepunktet vil være avslørende for om posisjonering stemmer med virkeligheten. Hvis roboten ender opp i samme punkt er det en indikasjon på god posisjonering.

Testen gjennomføres utelukkende med sensorfusjon av motorodometri (translasjon) og IMU (vinkel) som tilbakemelding til ruteplanlegger, grunnet resultat for de resterende relative sensorene. Testen logger derimot data fra alle relative sensorer, som vil presentere et bilde av hvordan sensoren *oppfatter* den tilbakelagte ruten.

4.3.2 Resultat

Test 1: punkt til punkt, motorodometri

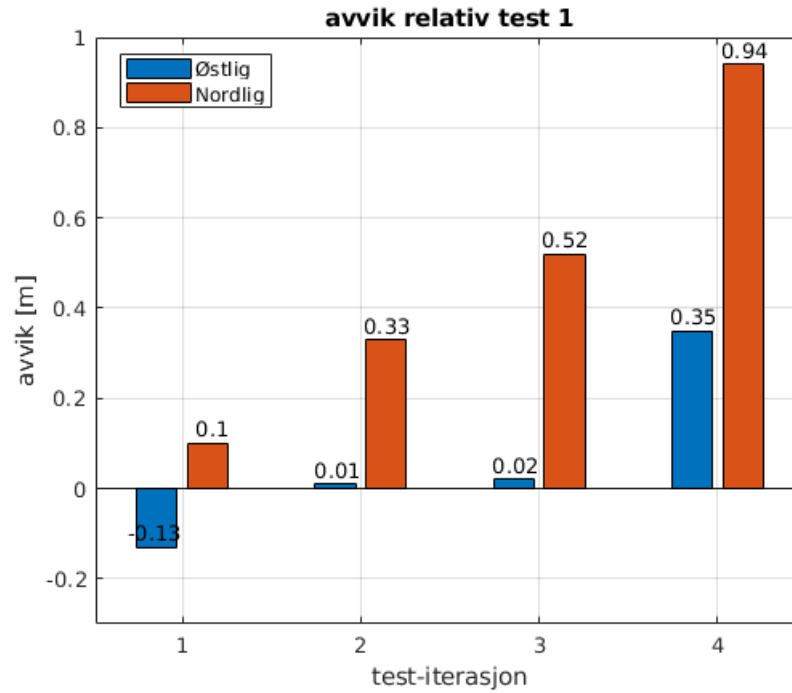
Figur 45 viser et forsøk på å kjøre til et punkt. Det ble forsøkt flere ganger å legge tilrette for å få et sammenlignbart resultat uten hell. Videre testing benytter seg ikke av denne konfigurasjonen.



Figur 45: Resultat, punkt til punkt, motorodometri

Test 1: punkt til punkt, sensorfusjon

Figur 46 viser avviket mellom faktisk stoppunkt og ønsket stoppunkt for alle gjennomføringer.

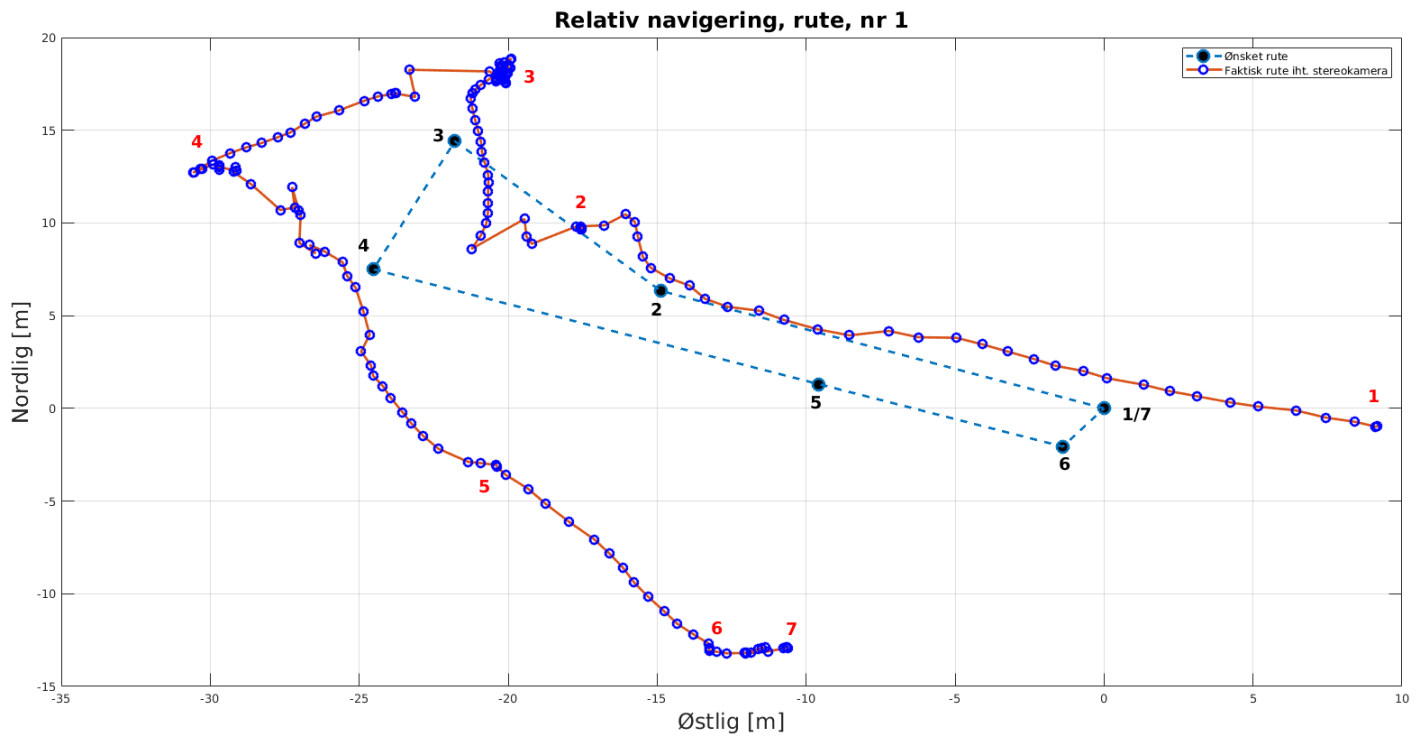


Figur 46: Resultat, relativ navigasjon test 1

Presisjonen er meget god ved første gjennomføring, men den blir gradvis dårligere for hver iterasjon.

Test 2: rute, stereokamera

Figur 47 viser stereokameraets oppfattelse av ruten.

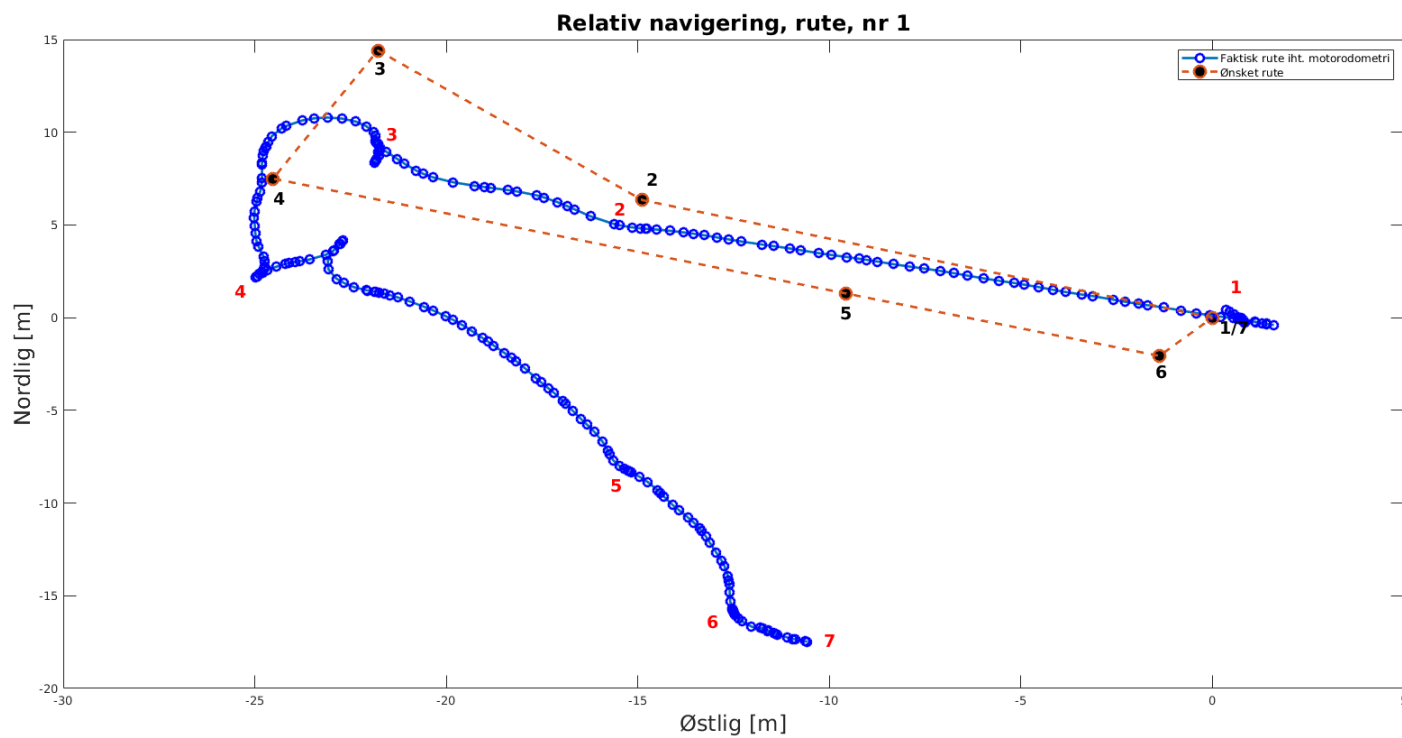


Figur 47: Rute iht. stereokameraets visuelle odometri

Resultatet viser at sensoren beskriver ruten unøyaktig i forhold til virkelig rute. Translasjon stemmer best, men ikke bra, mens estimert rotasjon stemmer svært dårlig.

Test 2: rute, motorodometri

Figur 48 viser motorodometriens estimat av tilbakelagt rute.

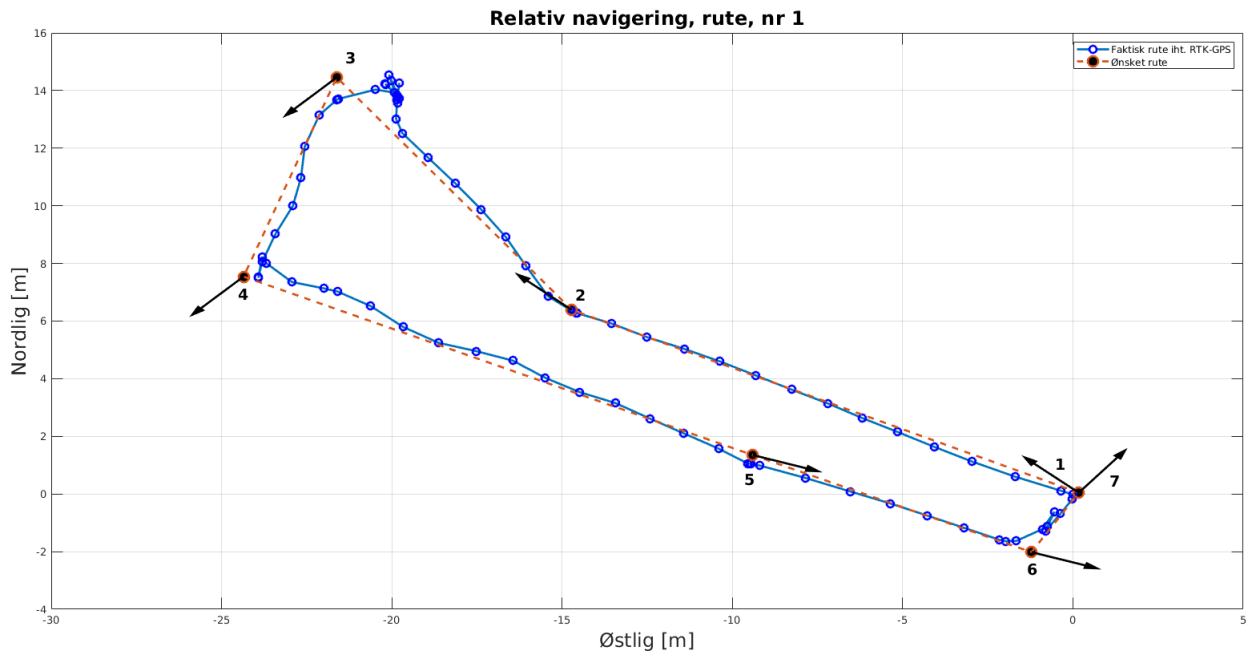


Figur 48: Rute iht. motorodometri

Avstand stemmer relativt bra mens retning er den største bidragsyter til feil.

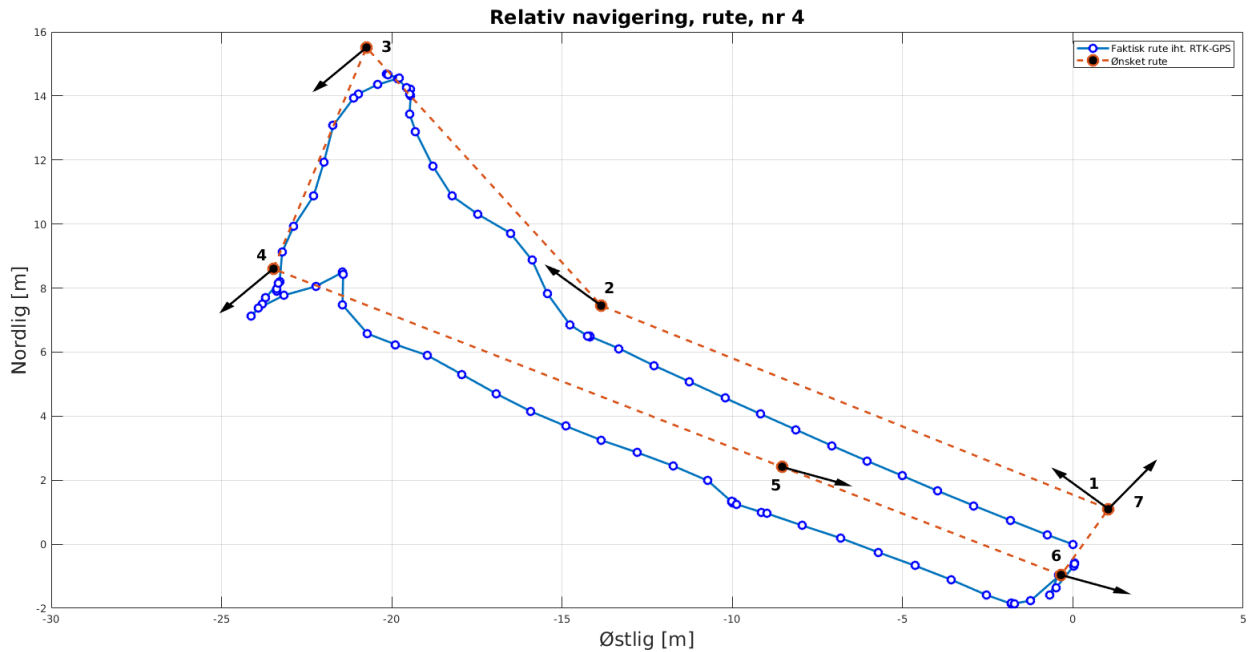
Test 2: rute, sensorfusjon

Figur 49 viser resultatet for første iterasjon av rute med relative sensorer.



Figur 49: Relativ, rute, gjennomføring 1

Med unntak av punkt nr. 3 treffer Spurv godt innenfor 0.3 m av rutepunktene. Den treffer også godt på siste punkt i ruten, som er en god indikator.



Figur 50: Relativ, rute, gjennomføring 4

Begrensingen til navigasjon med relative sensorer begynner å vise seg etter første gjennomføring. Figur 50 presenterer iterasjon nr. 4 som viser et tydelig statisk og voksende avvik i sør-vestlig retning.

4.4 Navigasjon ved global posisjonering

Kapitlet tar for seg navigasjon i globalt referansekoordinatsystem med fusjon av både global og relativ lokaliseringsdata som tilbakemelding til ruteplanlegger. Hensikten med testene i delkapitlet er å avdekke hvor presis den globale sensoren er, samt hvor god presisjon sensorfusjon av global og relative sensorer kan oppnå. Tanken er at den globale sensoren skal kompensere for feilen som akkumuleres i relative sensorer.

Alle testene gjennomføres også her med en måltoleranse på 0.3 m og ca 30°. RTK-basestasjonens presisjon er 0.26 m for alle tester.

4.4.1 Testoppsett

I de påfølgende avsnittene gjennomføres det tester med tre ulike oppsett. GNSS-mottaker refereres videre til som GPS-mottaker, men benytter også signaler fra satellitter i satellittnavigasjonssystemet GLONASS. For alle tester logges følgende informasjon for å presentere resultat:

- `Topic /fix` : Ubehandlet GPS- eller RTK-GPS-data direkte fra mottakerens ROS-driver.
- Fysiske målinger med målebånd.

Testene er beskrevet som følgende:

Test 1: GPS-presisjon

Hensikten med testen er å tallfeste GPS-mottakerens nøyaktighet. Den gjennomføres ved å la roboten stå i ro i et tidsrom på 10 minutter mens GPS-mottakerens posisjon logges. Testen har to ulike oppsett, hvor det første benytter kun GPS-mottaker og deretter med korreksjoner fra RTK-basestasjon.

Test 2: punkt til punkt

Den andre testens hensikt er å vurdere det globale tilstandsestimatets presisjon. For å kunne måle med så høy presisjon som praktisk mulig, benyttes landemerker som er synlig i tilgjengelig kartdata som referansepunkter. På denne måten kan et geografisk punkt beskrives med høy nøyaktighet i lengde- og breddegrader, samtidig som det enkelt kan lokaliseres i testområdet. Figur 51 viser benyttede referansepunkt.



Figur 51: Referansepunkt benyttet i tester for global posisjonering (vannavløp).

Testen går ut på å sende roboten fra punkt 1 til punkt 2 for deretter å måle avvik fra ønsket endeposisjon med målebånd. Testen repeteres fire ganger for å minske påvirkning av tilfeldigheter. Testen benytter også to sensoroppsett, som kartlegger nøyaktighet ved bruk av GPS og deretter RTK-GPS.

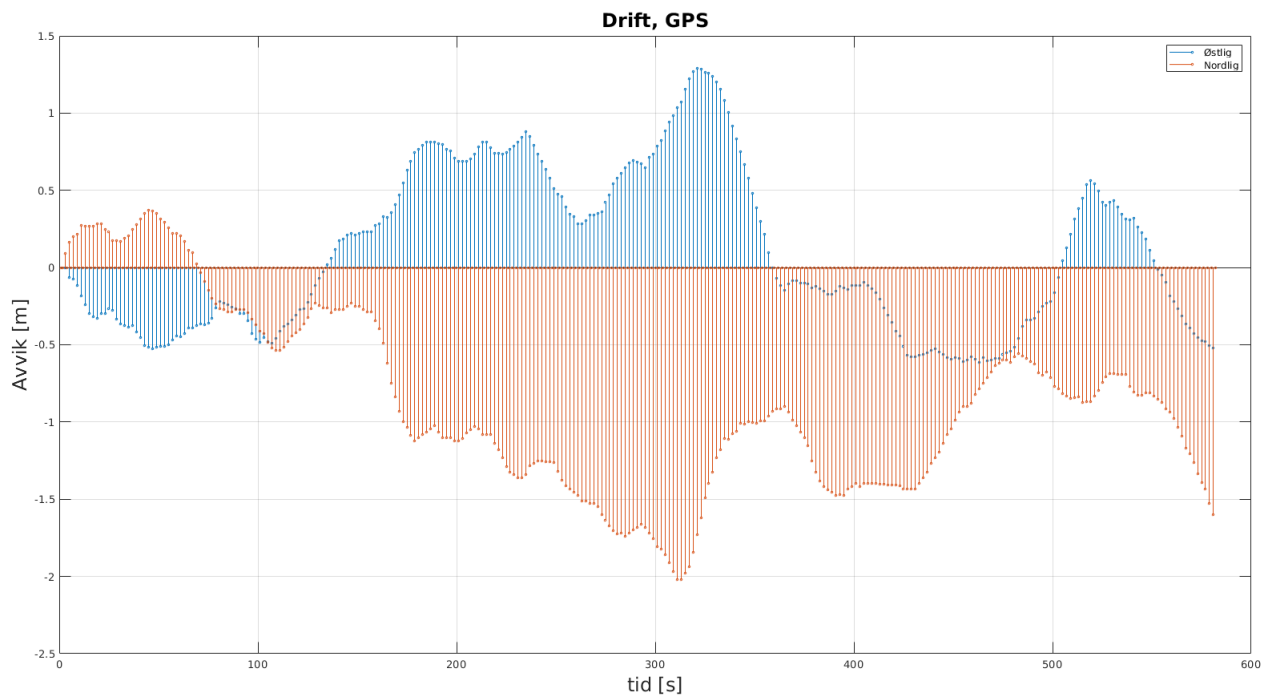
Test 3: rutefølgning

Testens formål er å avdekke hvordan det globale tilstandsestimatet påvirker robotens presisjon over en lengre distanse som også inneholder krappe svinger. Ruten er identisk med ruten benyttet for relativ navigasjon slik at de kan sammenlignes. Her er det interessant å finne ut om det statiske avviket påvist ved relative sensorer reduseres, og om nye feilkilder fra global sensor påvirker resultatet.

4.4.2 Resultat

Test1: GPS-mottakers presisjon

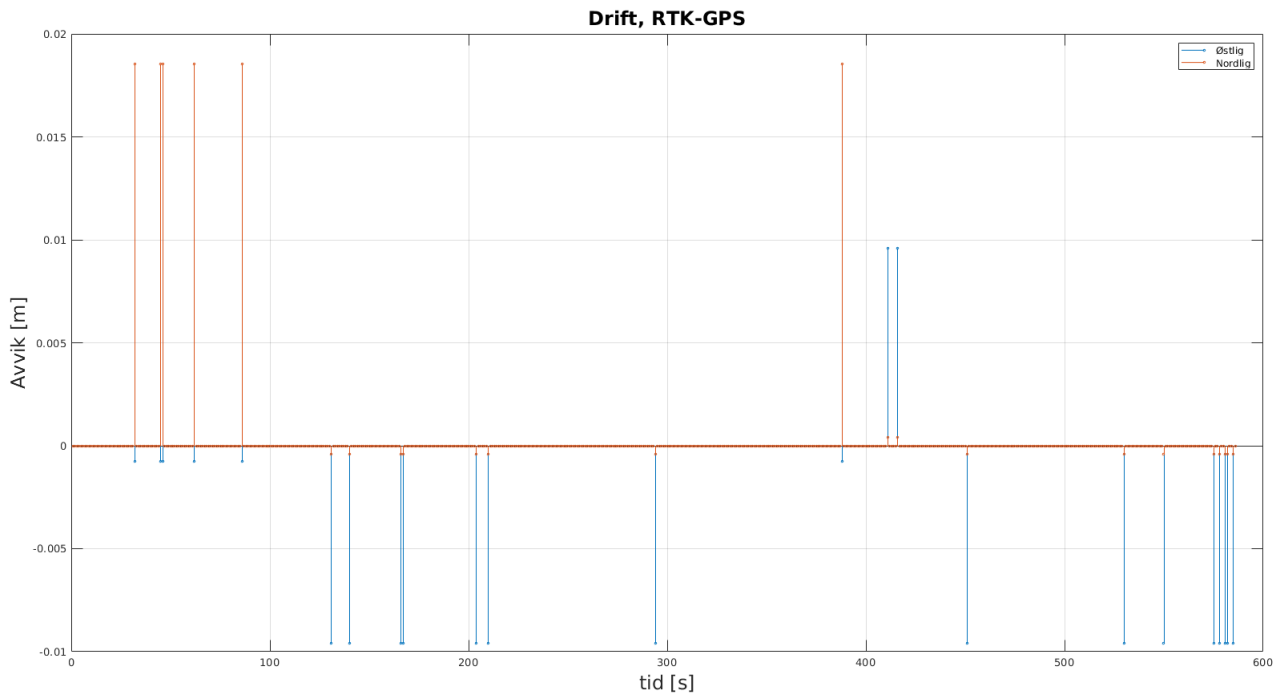
Figur 52 viser hvordan GPS-målinger uten korreksjonsdata utvikler seg over tid med roboten stående i ro. Grafen representerer her avvik i forhold til *første* måling og sier således ikke noe direkte om nøyaktighet, men om varians i sensorens estimat.



Figur 52: Figuren viser GPS-mottakerens presisjon over 10 minutter med roboten stående i ro.

Grafen viser at avviket varierer mellom 0 - 1.5 m i østlig retning, og 0 - -2 m i nordlig retning. Estimaterne avviker i begge retninger, og krysser nullpunktet flere ganger i løpet av testperioden.

Figur 53 viser hvordan GPS-målinger med korreksjonsdata fra RTK-basestasjon utvikler seg over tid

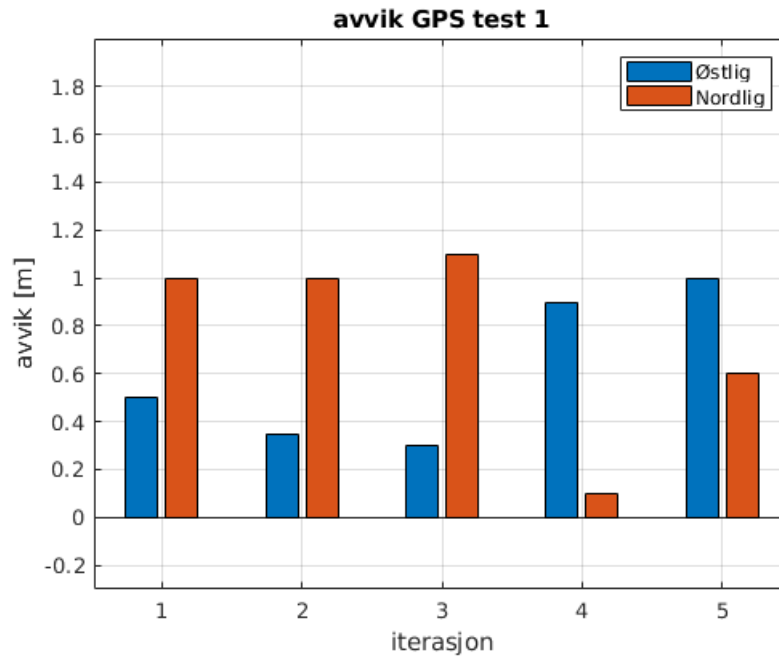


Figur 53: Figuren viser GPS-mottakers presisjon med RTK korreksjoner over 10 minutter. Basestasjonens presisjon er satt til 0.26 m. Legg merke til ulik oppløsning langs y-akse ift. forrige figur.

Figuren over viser at avvik er under målbar verdi under nesten hele testperioden. Enkelte avvik på 1-2 cm forekommer sjeldent.

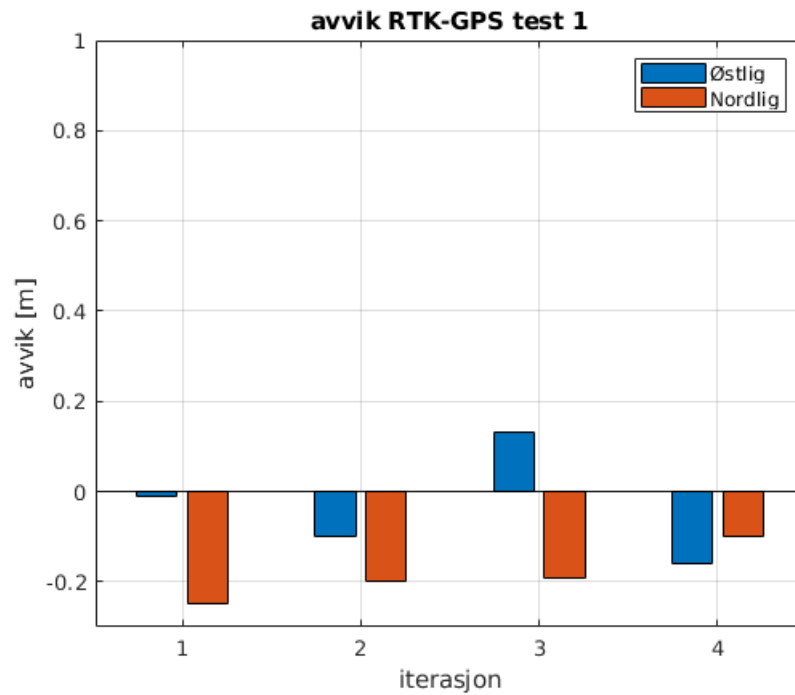
Test 2: punkt til punkt

Figurene i dette avsnittet presenterer avvik fra ønsket stopposisjon målt med målebånd.



Figur 54: Resultat, punkt til punkt med GPS-mottaker som global sensor.

For navigasjon med GPS-mottaker er avviket ved de 3 første iterasjonene størst i nordlig retning, mens for 4 og 5 er avviket størst i østlig retning.

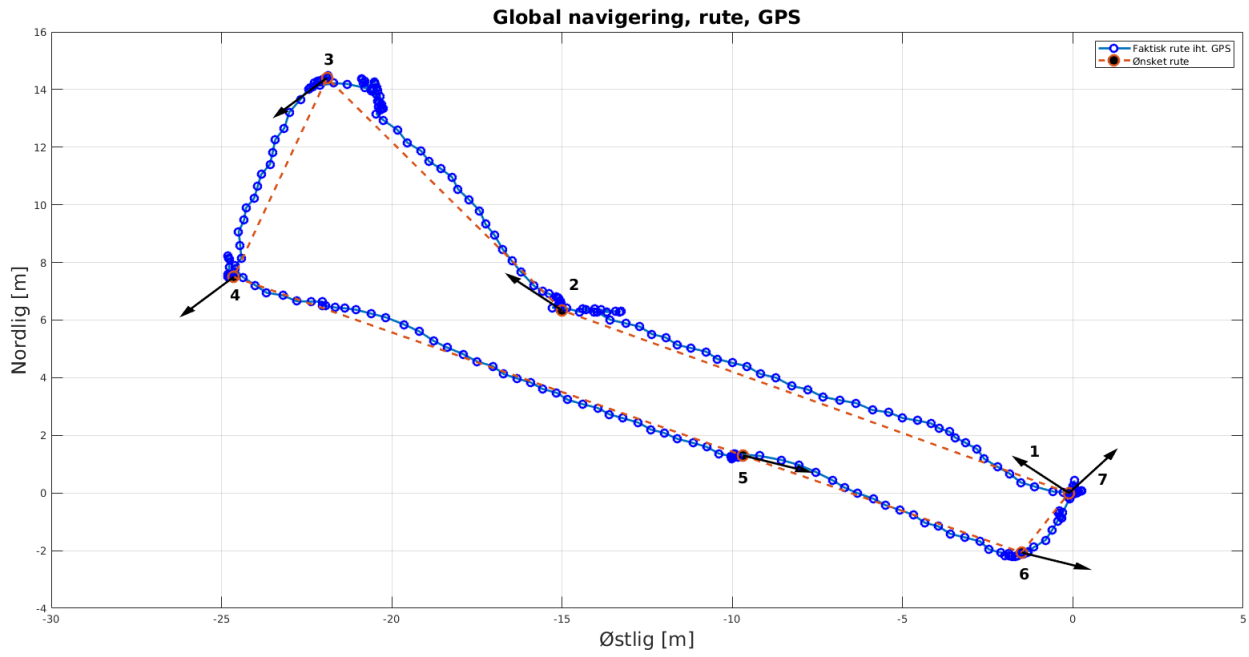


Figur 55: Resultat, punkt til punkt med GPS-mottaker og korreksjoner fra RTK-basestasjon som global sensor.

Figuren viser at avviket er mindre ved bruk av korreksjoner fra RTK-basestasjon, og holder seg innenfor satt nøyaktighet for basestasjon på 0.26 m.

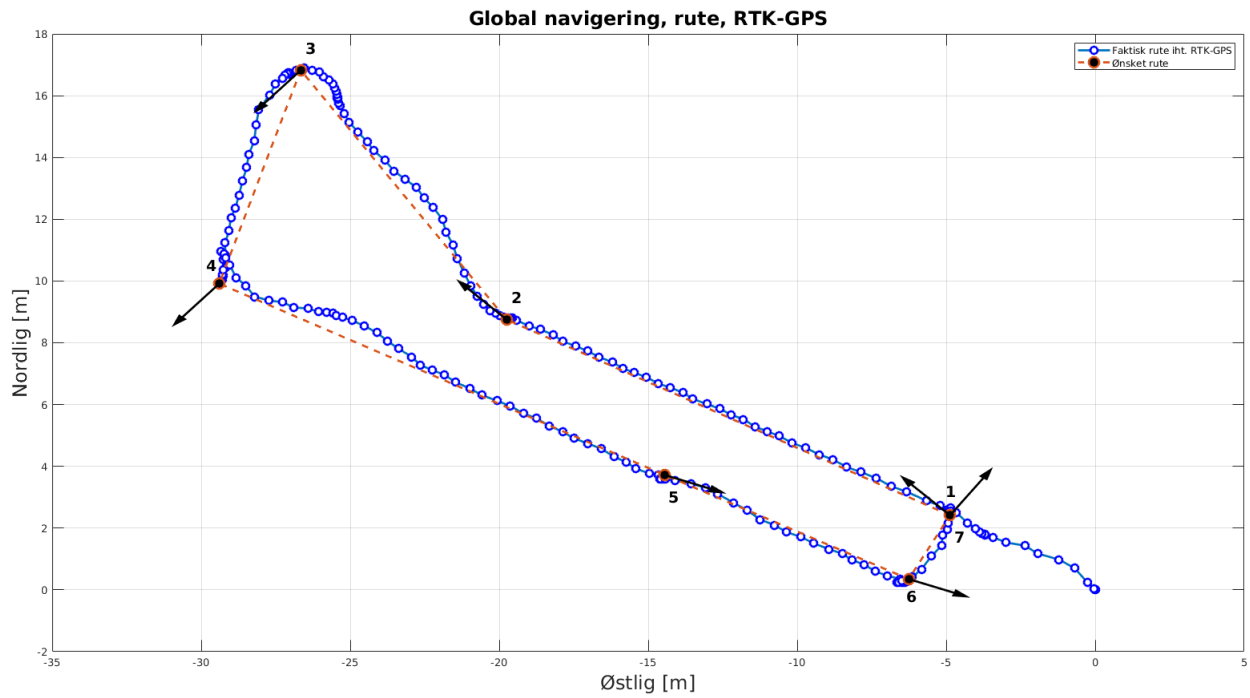
Test 3: Rutefølgning

Figurene under viser resultatet ved bruk av GPS-mottaker først, deretter med korreksjoner fra basestasjon.



Figur 56: Resultat rutefølgning, GPS: blå linje representerer robotens faktiske posisjon, hver markør på linjen representerer en punktprøve. Den brune linjen er korteste avstand mellom hvert veipunkt, her representert som brune sirkler. Sorte piler representerer ønsket yaw.

Figuren viser at roboten treffer innenfor 0.3 m av hvert punkt. Resultatet beholder tilsvarende presisjon ved gjentatte gjennomføringer uten akkumulering av feil fra relative sensorer.



Figur 57: Resultat rutefølging, RTK-GPS: Blå linje representerer robotens faktiske posisjon, hver markør på linjen representerer en punktprøve. Den brune linjen er korteste avstand mellom hvert veipunkt, her representert som brune sirkler. Legg merke til avstikkeren som går innom punkt 4. Sorte piler representerer ønsket yaw.

Figuren viser at roboten treffer godt innenfor 0.3 m av hvert punkt.

4.5 Modelling av omgivelser

Hensikten med testene i underkapitlet er å vurdere hvor nøyaktig stereokamera representerer nærmiljøet rundt roboten, samt hvordan hele systemet responderer ved en hindring som må forseres.

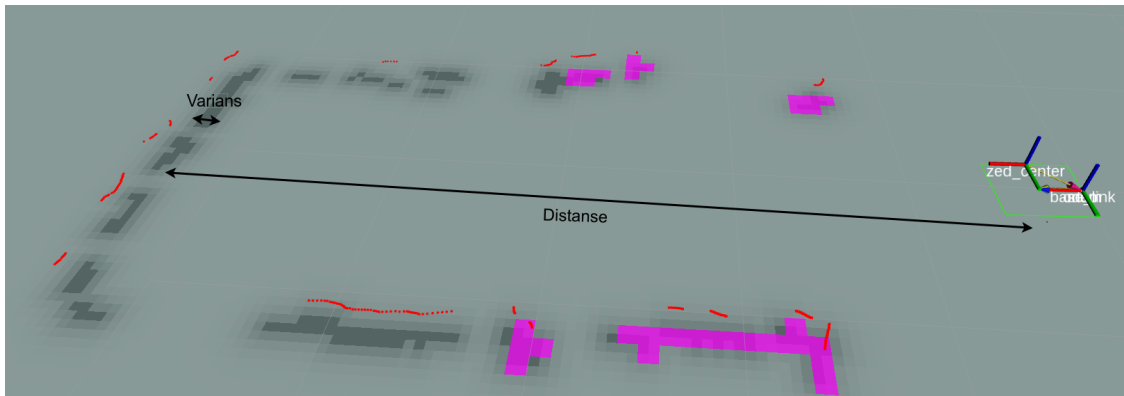
4.5.1 Testoppsett

For å vurdere stereokamera er det satt opp to tester som beskrevet under.

Test 1: Presisjon

Testen går ut på å peke roboten mot en vegg og deretter benytte visualisering fra `rviz` av `costmap` samt observasjoner fra sensor til å finne robotens estimat av distanse til vegg. Avstanden er først 1 m og økes gradvis opp mot 10 m. I tillegg noteres ”tykkelsen” eller varians for estimat, se figur 58.

Det antas at parameteren `confidence` for stereokamera må justeres noe underveis for at observasjoner skal bli lagt inn i `costmap`. Verdiens utgangspunkt er 70%.



Figur 58: Figuren viser testoppsett for vurdering av distanse til vegg, og indikerer hva som menes med størrelsene *distanse* og *variens*. Røde linjer representerer måling fra sensor.

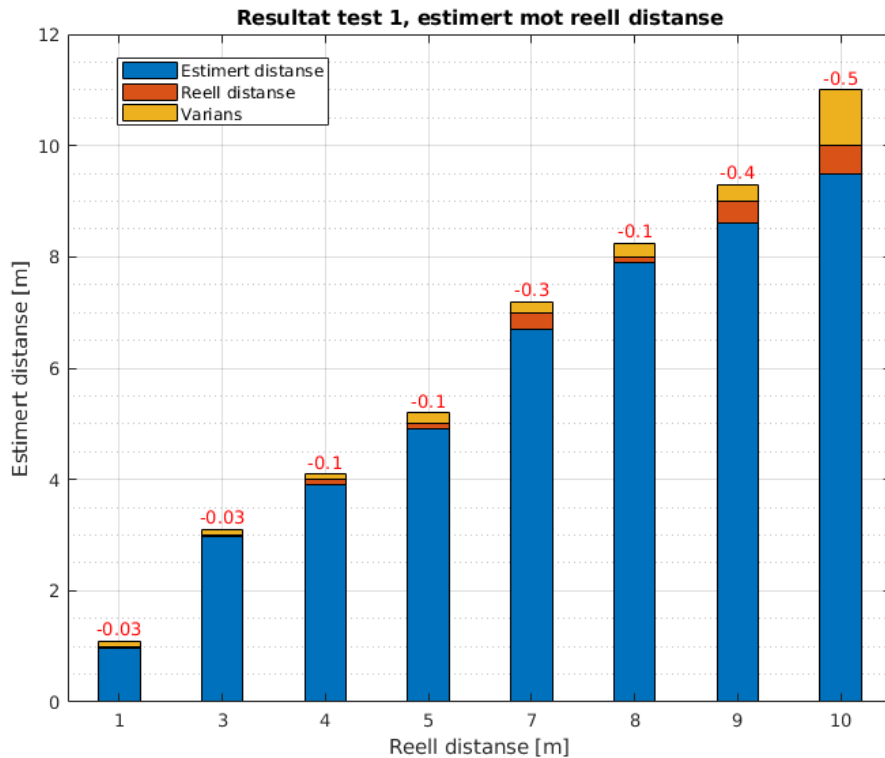
”Tykkelsen” på veggene forteller noe om sensordataens varians, mens distansen beskriver sensorens presisjon.

Test 2: Unngå hindring

Testen går ut på å kjøre roboten fra punkt til punkt som i navigasjonstestene, men nå med en hindring mellom start- og stopp-posisjon på ca 2 x 5 m.

4.5.2 Resultat

Test 1: Presisjon



Figur 59: Figuren viser stereokameraets estimerte distanse mot reell distanse. Blå farge er estimert distanse mens rødt representerer avvik ift. reell distanse, som også er skrevet i rødt over stolpen. Gul viser målingens tykkelse eller varians og er lagt oppå reell distanse.

Målingene ble gjort med **confidence** på 70% for avstand opp til og med 4 m, deretter 80% for 5 og 7 m og tilslutt 90% for 8-10 m. Uten justering av **confidence** viser kameraet ingen objekter over 4 m. Ved 10 m er målingen støyete og varierer med ca 1 m. Støy i målinger øker i takt med lavere **confidence**.

5 Diskusjon med konklusjon

Diskusjonskapitlet tolker resultatenes betydning og binder dem sammen med deloppgavene fra introduksjonen. Resultatene diskuteres i samme rekkefølge som de er presentert.

5.1 Evaluering av odometri

Hensikten med testene i de to avsnittene som omhandler odometri er å finne ut hvor presist de relative sensorene estimerer egen posisjon, samt å kalibrere eller justere dem for oppsettet i denne oppgaven. Dette inngår i løsningen for deloppgaven ”forbedring av posisjonsnøyaktighet”, som er et svært viktig aspekt for semi-autonom oppførsel.

5.1.1 Relative sensorers presisjon

Translasjon

Den første testen viser at den opprinnelige verdien for parameteren $K_{SpeedToRpm}$ for motorodometri gjorde at posisjonsestimats tilbakelagte distanse ble 20% for langt. Det vil i praksis si at hastigheten er ca 20% for lav, noe som ga relativt stort utslag ved testens hastighet på 1 m/s, men som ville vært enda mer betydelig ved større hastigheter. Etter utregning og justering av parameteren kommer feilen ned i 1%, noe som er bra, men det kan likevel ikke påstås at estimats feil er på 1% over større avstander. Testdistansen på 20 m er relativt kort og tillater testing under kontrollerte forhold, men som ikke nødvendigvis får fram feilen over lengre tidsrom. Som et resultat av det vil distansefeilen vokse sakte men sikkert slik at feil akkumuleres lineært med antall kjørte meter. Det er likevel viktig at denne sensoren kalibreres så bra som det lar seg gjøre, slik at estimats blir best mulig så lenge som mulig. Denne degraderingen er, som nevnt i oppgavens innledning, forventet. Den eneste måten å korrigere for oppbygging av slik feil er å benytte en form for *global* posisjoneringskilde.

En annen relativ sensor som ble testet er visuell odometri fra stereokamera. Den første testen viser at kameraet rapporterer posisjon med en feilmargin på bare 6%, som er betraktelig bedre enn motorodometriens 20%. Den viser det beste resultatet før justering av motorodometri, men har også den ulempen at estimats ikke kan justeres i form av parametre, slik som for motorodometri. Da den baserer seg på visuell data er den naturligvis også følsom for andre bevegelser i scenen, som forbi-passerende kjøretøy, men vil også kunne bli påvirket av lysforhold, spesielt sterkt sollys og mangel på lys nattetid. Det har en tendens til å produsere ulike resultater ved repetisjon av samme situasjon, i tillegg til feil som varierer i takt med endring av topografi underveis.

Posisjonsestimater fra sensorfusjon er som forventet tilnærmet likt som for motordometri, siden begge tross alt benytter data fra samme sensor. Forskjellen ligger i at IMU detekterer endring i yaw som påvirker fordelingen av tilbakelagt avstand i de to dimensjonene. Det medfører at kombinasjonen av sensorer klarer å oppdage ukjente feil i systemet som fører til vilkårlig avvik fra ønsket retning, som resulterer i at systemet klarer å kompensere for feilen. Basert på det utfallet konkluderes det med at IMU er en kritisk komponent for systemets funksjonalitet.

Sving

Som ved første vurdering av translasjon viser også resultatet for sving at motordometri helt klart hadde potensiale for forbedring. Parameteren $K_{Steering_to_Servo}$ sett i forhold til resulterende posisjonsestimater viste seg å ha en for høy verdi og dermed for høy beregnet radius, 140% og 73% over for hhv. venstre og høyre. Den opprinnelige parameterverdien tilsvarte en minimum svingradius på 1.6 m, mens reell minimum er nærmere 1 m. Eksperimentet viste også at det er forskjell i radius for høyre- og venstre-sving, som i praksis betyr at roboten svinger krappere eller skarpere til venstre. Det ble bestemt å sette en verdi som lå rett over den største minimumsverdien (høyresving var 0.94m) slik at verdien gjaldt for begge retninger. Testen med nye parametre viste en markant forbedring med ny feilmargen på 34% og 6%, men grunnet forskjell i utslag for venstre er den fremdeles relativt høy for venstresving.

Etter justering ble det gjennomført en ny test for å finne ut om justerte parametre stemmer overens over et større område av servomotorens vinkelutslag. Resultatet viste et avvik på hhv. 22% og 18% som er en økning for høyre side men nedgang for venstre. Reell svingradius var tilnærmet identisk. Det tyder på at svingdynamikken ikke er helt lineær og at problemet med ulik svingradius oppstår når utslaget nærmer seg maksimal verdi.

Årsaken til problematikken med svingradius er funnet å ligge i koblingen mellom servomotorenes tannhjul og overføringsleddet som overfører kraft fra servomotor til svingmekanismen. En justering av overføringsleddet ble forsøkt under arbeidet for oppgaven, uten hell. Løsning ligger sannsynligvis i å bytte servomotor eller overføringsledd slik at de passer overens.

Oppsummert kan det sies at det oppstår problemer ved å bruke servomotorens pådrag som grunnlag for beregning av odometri, spesielt når det er funnet ulineær oppførsel ved sving. Med årsaken i dette og forrige avsnitt til grunn anbefales det å benytte en servomotor som gir tilbakemelding om reell posisjon og som passer bedre med svingmekanismen på Spurv.

I etterkant av testene beskrevet over ble det lagt merke til at svingutslag ved automatisk navigasjon var betydelig mer forsiktig enn tidligere, noe som resulterte i at små svingvinkler ga et så lavt utslag at det i praksis betød ingen sving. Det skyldtes at parameteren $K_{Steering_to_Servo}$ ble nedjustert som resultat av testene over. Siden parameteren også bestemmer faktisk utslag for servomotor basert på ønsket svingvinkel,

er det en naturlig konsekvens. Konsekvensen underbygger i tillegg argumentet om å ikke benytte pådrag som tilbakemelding, og førte til at verdien måtte settes noe opp, til ca 1.0.

5.2 Navigasjon

5.2.1 Navigasjon ved relativ posisjonering

Motorodometri som tilbakekobling

Resultat for testgjennomføring med motorodometri viste som forventet et dårlig resultat, og hensikten med testen var å demonstrere systemet slik det var før ekstra sensorer ble installert. Figur 45 viser at det oppstår betydelige problemer ved forsøk på selvstendig navigasjon, og roboten kommer seg ikke til første punkt før den ble stoppet grunnet dårlig utvikling. Hovedkilden til den dårlige ytelsen er mangelfull tilbakemelding om tilstanden *yaw*, slik at roboten har stor usikkerhet angående hvilken retning den kjører og produserer derfor hyppige kursendringer. Det resulterer i en oppførsel med oscillerende svingutslag og en lav hastighet i mer eller mindre tilfeldige retninger.

Motorodometri

Figur 48 viser hvordan motorodometri oppfatter ruten. Gjennomføringen får frem at relative avstander stemmer godt overens med reell avstand, mens estimatet blir nok en gang degradert idet roboten svinger. Det er tydelig at estimat av vinkel kan forbedres med en sensor som *måler* vinkelendring.

Stereokamera

Figur 47 viser hvordan stereokameraets odometri oppfatter en runde rundt ruten. Den første delen av ruten er en rett bane og her stemmer estimatet relativt bra. Ved punkt 2 skjer det noe ukjent som "forstyrrer" kameraet, som gir et diskret hopp i både translasjon og *yaw*. Videre fortsetter den i omtrent samme retning til punkt 3, før et nytt diskret hopp forekommer. Resten av ruten stemmer omtrentlig, vel og merke med en tydelig feil verdi for *yaw*.

Testen som viser stereokameraets oppfattelse av ruten får godt frem problemene med visuell odometri. Underveis forekommer det diskrete hopp i både vinkel og translasjon, som kan forklares ved at programvaren for visuell odometri har problemer med å skille mellom segmenter i topografien rundt roboten som ligner på hverandre. Resultatet illustrerer også at både translasjon og rotasjon stemmer dårlig med virkeligheten. Differansen mellom start- og stopp-posisjon i resultatet er ca 12 m, mens i virkeligheten er den i verste fall noen få cm.

Resultatet for stereokamera står i kontrast til de innledende testene, som viser gode resultater for enkle testoppsett. Først ved mer avanserte testoppsett som kombinerer

lengre strekk og rotasjon, kommer svakhetene frem.

På bakgrunn av presisjon oppnådd gjennom tester for visuell odometri, konkluderes det med at sensoren ikke bidrar positivt til estimering av posisjon, og er derfor heller ikke benyttet i videre testing.

Sensorfusjon som tilbakemelding

Den siste testen i kapitlet for navigasjon ved relativ posisjonering benytter sensorfusjon av motorodometri og IMU som tilbakemelding til navigasjonssystemet. Resultatet viser at ved den første testgjennomføringen følger roboten ruten med høy presisjon, og treffer godt innenfor de 0.3 m som tilsvarer ruteplanleggers måltoleranse. Det tyder på at tidligere justering av motorodometri har fungert som ønsket, samt at IMU sitt estimat av absolutt vinkel stemmer godt med virkeligheten. Som forventet forekommer det likevel en gradvis akkumulering av feil som blir tydeligere for hver testgjennomføring. ”Formen” på estimert rute er likevel nærmest identisk for hver runde, som kan forklares med at *utgangspunktet* (punkt 1 i ruten) for start av ruten forskyves noe for hver gang, og dermed blir hele ruten tilsvarende forskjøvet. Dette fenomenet forekommer sannsynligvis fordi motorodometri ikke rapporterer presis nok translasjon, og dermed ”tror” roboten at den faktisk er i samme posisjon hver gang. En liten feil per gjennomføring vokser seg dermed større og større.

Slike problemer er typisk for sensorer som baserer seg på å måle tilbakelagt strekning fra et utgangspunkt uten en form for verifisering av posisjon. Avhengig av krav til presisjon vil et slikt system likevel gjøre nytte over et viss tidsrom, spesielt når andre posisjoneringsskilder ikke er tilgjengelige.

5.2.2 Navigasjon ved global posisjonering

Målsetningen ved å benytte GNSS som global sensor er å takle problemene som oppstår ved utelukkende bruk av relative sensorer. Her vurderes det i hvilken grad målet oppnås og hvor bra det resulterende systemet fungerer for posisjonering og navigasjon.

GNSS-mottaker som global sensor

For å vurdere presisjonen til systemet ved bruk av GNSS-mottaker, ble det gjennomført en test som målte avvik fra ønsket posisjon. Det kom fram at avviket er i størrelsesorden 0.3-1 m, som er relativt lavt.

En ulempe som oppstår ved bruk av GNSS i forhold til relative sensorer er at posisjonsestimaten kontinuerlig endrer seg innenfor en viss feilmargin. Feilmarginen ligger typisk på 1-3 m, som vil si at estimert posisjon vil endre seg i form av diskrete hopp innenfor 1-3 m. Bruken av Kalman-filter sørger for at hoppene er kortere ved høy usikkerhet, men å justere filteret for å *minimere* slike hopp vil også gå utover det endelige posisjonsestimats nøyaktighet. For å danne et bilde av i hvor stor

grad slike problemer forekommer ble det gjort en test hvor mottakerens estimat ble logget mens roboten var stillestående. Resultatet viste at avviket varierte mellom 0 - 1.3 m i østlig retning og 0 - 2 m i nordlig retning, samtidig som usikkerheten varierte mellom hhv. $0.9 - 1.3m^2$ og $1 - 1.5m^2$.

Vurdering av i hvilken grad avviket spiller en rolle kommer an på hvilket bruksområde systemet er tiltenkt. For et lite kjøretøy som Spurv kan avviket være av stor betydning, da avviket tilsvarer opptil 4 ganger kjøretøyets lengde, men det kan også tenkes at topografien er av en art som tilsier at avviket kan tolereres. En annen ulempe ved satellittnavigasjon er at mottakeren er avhengig av fri sikt til himmelen som fører til at estimatet degraderes i takt med antall tapte satellitter. Her kommer fordelen med sensorfusjon inn; ved tap av satellittdekning vil de relative sensorene fremdeles produsere et estimat som er relativt nøyaktig for en kort tidsperiode, og så lenge sikt til himmelen gjenopprettes innen et viss tidsrom (eller kort fysisk avstand) vil posisjonsestimaten fremdeles ha en lav feilmargin.

Med henvisning til oppgavens introduksjon, hvor det spesifiseres et område med *bebyggelse*, vil det resulterende systemet kunne takle ihvertfall korte perioder uten dekning.

GNSS-mottaker med RTK-korreksjoner som global sensor

For å forbedre posisjonering ytterligere ble det benyttet en GNSS-mottaker som har støtte for å benytte tilsendte korreksjoner fra en RTK-basestasjon. Oppnådd nøyaktighet i testen er i størrelsesorden 0 - 0.25 m, som er et svært godt resultat. Det tilsvarer maksimalt halvparten av kjøretøyets lengde, noe som vil være god nok presisjon for de aller fleste situasjoner. Tilsvarende test for RTK ved stillestående robot viser at avviket varierer svært lite; det er 0 mesteparten av tiden og er oppe i 0.01-0.02 m enkelte ganger. Det betyr at ikke bare har posisjonsestimaten høy presisjon, det er også meget stabilt.

Under konstruksjon og testing av programvare for rutefølgning ble det gjort en viktig, men sett i etterkant kanskje innlysende, observasjon ved navigasjon med global posisjonering. Når roboten mottar en rute bestående av flere rutepunkter er det viktig at transformasjon av rute fra globalt til robotens koordinatsystem foregår underveis, helst så tett opp mot aktuelt punkt som mulig, og ikke i forkant. Blir det derimot gjort i forkant vil de resulterende koordinatene være korrekte idet roboten starter rutefølgning, mens etter en viss tid/distanse vil transformasjonen `map` \rightarrow `odom` endre seg for å kompensere for oppdaget feil, og dermed flyttes også referansekoordinatsystemet `odom` som målkoordinatene beskrives i forhold til. Problemet unngås ved å ikke utføre transformasjon av koordinater før målet skal navigeres til. Det kan også spekuleres i om problemet også vil dukke opp ved stor distanse mellom rutepunktene.

Ved oppgaver i områder hvor det er ønskelig med bedre presisjon enn én meter, vil GNSS-mottaker med RTK-basestasjon gi meget god presisjon. Ulempen med å ta i bruk et slikt system er at det krever en statisk basestasjon og at roboten får

en begrensning i mulig operasjonsområde da den må befinne seg innen fri sikt av basestasjonen. Basestasjonen er også avhengig av en oppstartstid på flere timer, med mindre posisjonen den står i er kjent og kan beskrives med koordinater med høy presisjon. På en annen side er det ikke usannsynlig at det i nær fremtid er plassert ut RTK-basestasjoner, spesielt i større byer. Selskapet som produserer mottakeren tatt i bruk i denne oppgaven har gjennomført et slikt prøveprosjekt. Det er også teoretisk mulig å sende korreksjoner over andre databærere som 4G som kan sende korreksjoner basert på omtrentlig lokasjon.

Oppgaver

I konstruksjonskapitlet beskrives tilleggsfunksjonalitet som er tilført noden for rutefølgning, i form av mulighet for å starte video-opptak og spesifisering av retning og tidsrom for inspeksjon av objekter. Det er i oppgaven ikke gjennomført dedikerte tester som demonstrerer denne funksjonaliteten, grunnet prioritering av andre tester.

5.2.3 Deteksjon og unngåelse av hindringer

Presisjon

Presisjonen presentert i den første testen viser at stereokamera fungerer godt til å bedømme distanse til objekter opp til 10 m. Resultatene er riktignok produsert under kontrollerte forhold med roboten stående i ro, med godt lys samt at objektet er stort og befinner seg midt i kameraets synsfelt. I tillegg justeres **confidence** (fra 70%) slik at det mulig å oppdage objekter for større avstander enn ca 4 m (til 80-90%). Ulempen med denne justeringen er at objekter ved korte distanser inneholder mer støy jo høyere verdi, som resulterer i et ”rotete” **costmap** med falske objekter ved medium til høye distanser. Det bør derfor generelt benyttes en så lav **confidence** som mulig, og 70% er funnet å være et godt kompromiss mellom identifiserbar distanse og støy.

Unngåelse av hindringer

Testen som vurderer systemets evne til å oppdage og unngå hindringer viser at det fungerer tilfredsstillende.

I etterkant er det tydelig at størrelsen på lokalt **costmap** har spilt en rolle i årsaken til at roboten kjører såpass nært hindringen før den realiserer at den må rygge litt for å kjøre rundt. Størrelsen var under testen 5.5 m, som betyr at lokal ruteplanlegger kun tar hensyn til objekter 2.75 m foran seg. Kombinert med litt tid til prosessering og hastighet rett mot hindring resulterte det i en stans på ca 1 m foran, som er akseptabelt, men i grenseland. Oppførselen videre i testen demonstrerer systemets fleksibilitet ved at det tar nødvendige hensyn basert på tilgjengelig informasjon, og kommer seg rundt et relativt stort hinder uten problemer.

Gjennom oppgavens utviklingsperiode er det gjort flere tester som ikke kommer med

her, men som har vist at systemet takler både små og store hindringer på en god måte. Selv om sensoren har en tendens til å produsere falske objekter i tillegg til mangelfull informasjon ved større distanser (>8 m) klarer systemet å fjerne de falske og legge til mangelfull informasjon når roboten nærmer seg objektene. Det observeres også at størrelsen på synsfeltet er bra og stemmer godt med teoretisk synsfelt basert på kameraets synsvinkel. Ved medium distanser (ca 3 m) degraderes likevel sensorens evne til å vurdere distanse i ytterkant av bildet, men fungerer godt under, slik at når roboten kjører nært langs en stor hindring ser den tydelig enden hvor den kan svinge inn for å komme seg rundt. Kameraet har også en begrensning i form av minimum avstand til objekt på 0.5 m. Det kan virke ubetydelig, men ved situasjoner der roboten havner innen 0.5 m av en hindring, for eksempel ved dynamiske hindringer eller ved kjøring i trange omgivelser, vil hindringen være ”usynlig” hvis den ikke er oppdaget på forhånd. Det kan medføre uønskede situasjoner som er vanskelig å gjøre noe med.

Avhengigheten av å ha oversikt frem i tid, og dermed distanse, øker i takt med robotens hastighet. En stor fordel er at robotens evne til å stanse er veldig god, som medfører at den kan stoppe på ca 0.5 m ved en hastighet på 8-10 m/s. For situasjoner hvor det er ønskelig at roboten skal operere med større hastigheter enn ca 7 m/s, kan det oppstå situasjoner hvor roboten ikke ser langt nok som vil medføre en kollisjon, på bakgrunn av en sikker synsrekkevidde på ca 8 m og en reaksjonstid på 1 s. Erfaringer viser også at justering av parametre for `costmap` er av stor betydning for navigasjon, slik at objekter representeres med en fornuftig men ikke for høy utvidelse. I tillegg nevnes det at robotens modell, både for `costmap` og ruteplanlegger, bør stemme godt med virkeligheten for god ytelse i trange omgivelser. Dersom slike parametre, grundigere forklart i kapittel 3, er fornuftig innstilt og `costmap` får god informasjon fra visuell sensor og dermed beskriver omgivelsene med god nøyaktighet, ligger alt til rette for at ruteplanlegger kan gjøre en god jobb. Den største begrensningen for navigasjon viser seg dermed å være stereokameraets evne til å beskrive omgivelser. For å oppnå bedre deteksjon ved medium til store avstander vil sannsynligvis en sensor med aktiv utsendelse (lys, ultralyd) yte bedre. Den vil heller ikke lide av problemer ved dårlige lysforhold eller minimums-rekkevidde.

5.3 Anbefalt systemoppsett

Basert på diskusjon av resultat fra tester gjennomført i denne oppgaven anbefales det følgende systemoppsett:

- Sensorfusjon hvor følgende sensorer inngår:
 - Relative sensorer
 - * Translasjon: motorodometri
 - * Vinkel: IMU
 - Global sensor
 - * GNSS-mottaker med RTK-basestasjon hvis praktisk mulig.
 - Maskinsyn
 - * Stereokamera fungerer tilfredsstillende til oppdagelse av hindringer på kort til medium avstand, men det anbefales å vurdere alternativ sensor med aktiv utsendelse.

I tillegg inngår det i oppsettet å følge anbefalte verdier for parametre som gjennomgås i kapittel 2.5 ”Parametre med særlig betydning”.

5.4 Videre arbeid og forbedringer

Systemet har flere aspekter som kan forbedres.

- **Metoder**
 - **SLAM** (Simultaneous Localization And Mapping): *norsk: samtidig lokalisering og kartlegging* er en prosess som benytter maskinsyn til å gjenkjenne tidligere besøkte områder, samtidig som den lagrer observasjoner som benyttes til å opprette et kart. Metoden er derfor å regne som en lokaliseringssensor i kjente områder. Det er under prosjektet utført enkle eksperimenter med SLAM, men grunnet begrenset kvalitet på data fra stereokamera, blir resultatet såpass degradert at det ikke gjør noen nytte. En slik lokaliseringsskilde regnes som en global sensor, og vil dermed komme til god nytte ved tap av satellittdekning for kompensering av relative feilkilder.
- **Sensorer**
 - *Mer presist maskinsyn*: Stereokamera benyttet i denne oppgaven fungerer tilfredsstillende til deteksjon av hindringer, men er ikke godt egnet til å kartlegge områder. En sensor som benytter aktiv utsendelse, f.eks vha. en

laser, antas å kunne gjengi fysiske objekters form og distanse fra robot på en mer presis måte, spesielt ved større avstander. Det vil også muliggjøre at roboten kan forflytte seg med større hastighet mens den tar hensyn til hindringer.

- *Maskinsyn akterover*: Det vil være fordelaktig å benytte en sensor som ser bakover slik at roboten får bedre oversikt over sitt nærmiljø. Det vil også forhindre situasjoner hvor roboten rygger inn i ukjente hindringer. En sensor som ser 360°, f.eks en **Solid State LIDAR** (*norsk: roterende lasersensor uten mekaniske deler*), vil gi svært god oversikt over områder, men er til gjengjeld kostbar.
- *Servomotor*: Dersom det skulle være ønskelig å ha en ekstra kilde for måling av **yaw**, må det installeres servomotorer som gir tilbakemelding om faktisk svingutslag. Det vil også sørge for mer nøyaktig *ønsket* svingradius.

- **Videreutvikling av autonom oppførsel**

- *Kompleks oppgavestyring*: Det finnes biblioteker i ROS som tillater å definere komplekse strukturer av oppgaver som kan inngå som en del av et autonomt system. Eksempler på slike biblioteker er **Smach** [40] og **ROS Commander** [41].
- *Maskinlæring*: For å få maksimalt utbytte av sensorene ombord på Spurv kan maskinsyn kombineres med maskinlæring for å kjenne igjen viktige gjenstander eller hendelser. Eksempler kan være branner, tilstedeværelse av mennesker i uønskede områder og deteksjon av våpen.

- **Annet**

- **Nettverk**: For å kunne støtte mange eksisterende nettverk, vil en enkel løsning være å introdusere støtte for trådløse nettverk som opererer i frekvensbandet 2.4 GHz. Dette er spesielt nyttig for utvikling ved skoler og universiteter hvor det er infrastruktur som dekker store områder.

5.5 Konklusjon

Rapporten viser at for å oppnå et velfungerende selvstendig navigasjonssystem er det av avgjørende betydning at roboten har tilgjengelig et godt estimat av *yaw*-vinkel, og at en IMU er i stand til å produsere et slikt estimat mens den innebygde motorodometrien ikke er det. Den fastslår også at bruken av visuell odometri er problematisk og at tilgjengelig stereokamera ikke produserer godt nok estimat av verken translasjon eller rotasjon for automatisk navigasjon.

For oppdagelse av objekter er stereokamera funnet å fungere tilfredsstillende.

Rapporten viser også resultater fra implementasjon av en global sensor for forbedring av posisjonsestimater med relative sensorer, og det konkluderes med et anbefalt oppsett basert på *sensorfusjon* av både de relative sensorene motorodometri og IMU, samt en global sensor basert på satellittnavigasjon for posisjonering.

Utfra definisjonen i introduksjon, kan systemet sies å inneha grunnleggende *semi-autonom* funksjonalitet, og er således istand til å følge en rute spesifisert av operatør samt beskytte seg selv mot påkjøring av objekter. Det foreslåtte systemet kan sees på som en plattform for videre utvikling av autonom oppførsel, og kan benyttes både av næringsliv og ved universiteter.

6 Referanser


- [1] Edson Prestes, Joel Luis Carbonera, Sandro Rama Fiorini, Vitor A. M. Jorge, Mara Abel, Raj Madhavanb, Angela Locoroc, Paulo Goncalves, Marcos E. Barretof, Maki Habibg, Abdelghani Chibani, Sébastien Gérardi, Yacine Amirat, Craig Schlenoffj, "Robotics and Autonomous Systems 61 (2013): Towards a core ontology for robotics and automation", Elsevier, side 1200.
- [2] Spesifikasjon, Nvidia Jetson TX2
<https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/>
- [3] Utgangspunkt for foroverkinematikk i kildekoden for noden vesc_to_odom
https://github.com/mit-racecar/vesc/blob/master/vesc_ackermann/src/vesc_to_odom.cpp
- [4] REP 103: Standard Units of Measure and Coordinate Conventions
<http://www.ros.org/repos/rep-0103.html>
- [5] REP 105: Coordinate Frames for Mobile Platforms
<http://www.ros.org/repos/rep-0105.html>
- [6] NMEA-navsat-driver kildekode fra github
https://github.com/ros-drivers/nmea_navsat_driver/pull/46/files
- [7] U-blox C94-M8p User Guide
https://www.u-blox.com/sites/default/files/C94-M8P-AppBoard_UserGuide_%28UBX-15031066%29.pdf
- [8] Artikkel om RTK funksjonalitet
<https://www.novatel.com/an-introduction-to-gnss/chapter-5-resolving-errors/real-time-kinematic-rtk/>
- [9] Bilde av Zed stereokamera fra www.stereolabs.com
<https://cdn.stereolabs.com/docs/overview/getting-started/images/ZED-Camera.png>
- [10] Artikkel om move_base fra wiki.ros.org
http://wiki.ros.org/move_base
- [11] Bilde av noder og topic fra wiki.ros.org
http://wiki.ros.org/custom/images/wiki/ROS_basic_concepts.png
- [12] Artikkel om ROS-noder fra wiki.ros.org
<http://wiki.ros.org/Nodes>
- [13] Artikkel om ROS-meldinger fra wiki.ros.org
<http://wiki.ros.org/msg>

-
- [14] Artikkel om ROS-topic fra wiki.ros.org
<http://wiki.ros.org/Topics>
 - [15] Artikkel om ROS-services fra wiki.ros.org
<http://wiki.ros.org/Services>
 - [16] Artikkel om ROS-master fra wiki.ros.org
<http://wiki.ros.org/Master>
 - [17] Artikkel om ROS-bags fra wiki.ros.org
<http://wiki.ros.org/rosbag>
 - [18] Artikkel om WGS84 fra wikipedia.org
https://en.wikipedia.org/wiki/World_Geodetic_System
 - [19] Bilde av oppdeling av lengde- og bredde-grad fra wikipedia.org CC0
https://commons.wikimedia.org/wiki/File:Latitude_and_Longitude_of_the_Earth.svg
 - [20] Bilde av UTM-soner fra wikipedia.org CC0
<https://commons.wikimedia.org/wiki/File:LA2-Europe-UTM-zones.png>
 - [21] Artikkel om UTM fra wikipedia.org
https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system
 - [22] Artikkel om NED fra wikipedia.org
https://en.wikipedia.org/wiki/North_east_down
 - [23] Bilde av NED-koordinatsystem fra wikipedia.org CC0
https://en.wikipedia.org/wiki/File:ECEF_ENU_Longitude_Latitude_relationships.svg
 - [24] Bilde som viser transformasjon fra lengde-og bredde-grader til odom fra ros.org
http://docs.ros.org/kinetic/api/robot_localization/html/_images/figure1.png
 - [25] Robot localization, samling av noder for sensorfusjon
http://wiki.ros.org/robot_localization
 - [26] "A Generalized Extended Kalman Filter Implementation for the Robot Operating System" av T. Moore and D. Stouch, 2014.
https://link.springer.com/chapter/10.1007/978-3-319-08338-4_25
 - [27] Artikkel om kvaternioner
<https://en.wikipedia.org/wiki/Quaternion>
 - [28] Tabelldata hentet den 07.04.18
<http://www.chrobotics.com/library/accel-position-velocity>
 - [29] Artikkel om costmap på ros.org
http://wiki.ros.org/costmap_2d
 - [30] Timed elastic bandslokal ruteplanlegger fra wiki.ros.org
http://wiki.ros.org/teb_local_planner

-
- [31] Global planlegger fra ros.org
http://wiki.ros.org/global_planner
 - [32] Artikkel som beskriver fenomenet "gimbal lock" fra wikipedia.org
https://en.wikipedia.org/wiki/Gimbal_lock
 - [33] Funksjonen "LL2UTM" er hentet fra ROS-pakken robot-localization i header-fil navsat_conversions.h
https://github.com/cra-ros-pkg/robot_localization/blob/kinetic-devel/include/robot_localization/navsat_conversions.h
 - [34] Matlab-funksjon "LLtoUTM", Francois Beauducel
<https://se.mathworks.com/matlabcentral/fileexchange/45699-ll2utm-and-utm2ll>
 - [35] ZED Stereokamera, teknisk spesifikasjon
<https://www.stereolabs.com/zed/>
 - [36] ROS-driver for IMU 3dm-gx4-25, github.com
https://github.com/KumarRobotics/imu_3dm_gx4
 - [37] Noden depthimage_to_laserscan fra wiki.ros.org
http://wiki.ros.org/depthimage_to_laserscan
 - [38] ROS-pakke costmap_converter, fra wiki.ros.org
http://wiki.ros.org/costmap_converter
 - [39] Kartdata fra google.com
<https://maps.google.com>
 - [40] SMACH, oppgavestyring for ROS
<http://wiki.ros.org/smach>
 - [41] ROS commander, oppgavestyring for ROS
<https://ieeexplore.ieee.org/document/6630616/>

7 Vedlegg

Vedleggene er vedlagt i PDF-dokumentet og kan åpnes ved å trykke på knappen til høyre for beskrivelsen.

- Kildekode for `gps_conversion_node.cpp` og `move_base_interface.py` 
- Parameter-dump som lister alle benyttede parametre i ROS 